HERRAMIENTAS PARA LA ENSEÑANZA DE SISTEMAS INTELIGENTES UTILIZANDO PLATAFORMAS DE HARDWARE LIBRE

CRISTHIAN DAVID CORDOBA PARRA
JESSICA LEANDRA VELASCO PEREZ

UNIVERSIDAD AUTÓNOMA DE OCCIDENTE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE AUTOMÁTICA Y ELECTRÓNICA
PROGRAMA DE INGENIERÍA MECATRÓNICA
SANTIAGO DE CALI
2017

HERRAMIENTAS PARA LA ENSEÑANZA DE SISTEMAS INTELIGENTES UTILIZANDO PLATAFORMAS DE HARDWARE LIBRE

CRISTHIAN DAVID CORDOBA PARRA
JESSICA LEANDRA VELASCO PEREZ

Proyecto de Grado para optar al título de Ingeniero Mecatrónico

Director
JESÚS ALFONSO LÓPEZ
Doctor en Ingeniería

UNIVERSIDAD AUTÓNOMA DE OCCIDENTE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE AUTOMÁTICA Y ELECTRÓNICA
PROGRAMA DE INGENIERÍA MECATRÓNICA
SANTIAGO DE CALI
2017

Nota de Aceptación:

Aprobado por el Comité de Grado en cumplimiento de los requisitos exigidos por la Universidad Autónoma de Occidente para optar el título de ingeniero Mecatrónico

Jimmy Tombe

Jurado

Juan Carlos Mena

Jurado

Santiago de Cali, 29 de Marzo de 2017

AGRADECIMIENTOS

A Dios, por mantenerme firme y no decaer a pesar de las adversidades, presentadas durante este gran esfuerzo y dedicación que comprendió la carrera Ingeniería Mecatrónica.

A mis padres, por su apoyo incondicional, su paciencia y entendimiento durante todos los años.

A mi abuela Lilia, por haberme aportado todo desde pequeño, estar siempre ahí, dándome entusiasmo para salir adelante y ser el soporte para lograr cada una de mis metas.

A mi director Jesús López, que sin su ayuda, gestión y conocimiento no hubiese sido posible realizar este proyecto.

A todas las personas que de una u otra forma colaboraron en la realización del presente proyecto.

CONTENIDO

LISTA DE TABLAS

LISTA DE ILUSTRACIONES

pág.

LISTA DE ANEXO

RESUMEN

En este proyecto se presentan diferentes aplicaciones de redes neuronales y de control difuso implementadas por medio de plataformas de hardware libre como Arduino y Microview. En total se implementaron cinco aplicaciones o herramientas. A cada una se le desarrolló su respectiva guía, donde se hace una descripción detallada de los materiales utilizados, de los diagramas esquemáticos, de sus respectivos códigos y de los resultados obtenidos. Las cinco herramientas implementadas cubren un amplio espectro de aplicaciones de las redes neuronales y del control difuso. La primera se enfoca en el reconocimiento de caracteres de tipo numérico (0-9), que mediante de *dip-switch* emula las posibles entradas que puede tener la red de tipo Perceptrón simple para el respectivo reconocimiento del carácter ingresado. La segunda práctica, consiste en el reconocimiento predominante de los colores rojo, verde y azul la cual por medio de *dip-switch*, emula las posibles entradas que puede tener la red de tipo Perceptrón multicapa para el respectivo reconocimiento del color predominante. La tercera consiste en un control difuso de temperatura, implementando como sensor de esta variable el LM35. La cuarta consiste en la visualización de una red neuronal, en un módulo basado en Arduino OLED llamado Microview, en el cual se puede verificar los valores de las entradas, los pesos, la neta y la salida, dependiendo de la función de activación seleccionada. Y la quinta consiste en la generación de señales Seno, Seno-Coseno en un Arduino mediante la implementación de un Perceptrón multicapa usando el mismo enfoque, se realizó la emulación de una planta de primer orden.

Palabras Clave: redes neuronales. Arduino. Microview. Control difuso. Perceptron multicapa. LM35.

INTRODUCCIÓN.

En el área de sistemas inteligentes existe una gran variedad de aplicaciones en diferentes campos orientados a solucionar diversos problemas, adaptándose de manera acorde al contexto presentado, permitiendo además una comprensión sintáctica y semántica del problema y su solución. Los sistemas inteligentes han presentado una evolución constante a lo largo de la historia, comenzando desde la aplicación más simple de una red neuronal como lo puede ser una red neuronal tipo perceptrón, hasta su expresión más compleja como lo es el *Deep Learning*, pasando además por la extensa aplicabilidad que permite la lógica difusa en el control de diversos dispositivos de uso cotidiano.[1]

Las redes neuronales artificiales conocidas como RNA son modelos que imitan el funcionamiento de las neuronas biológicas del cerebro. Este modelo es altamente simplificado en algunos aspectos por ejemplo en la emulación de la sinapsis y en la manera como se activa la salida de la neurona, pero en otra conserva una fuerte relación con su contraparte biológica como por ejemplo en la alta conectividad que tienen las neuronas biológicas y en la naturaleza distribuida del aprendizaje.

La lógica difusa es una manera de imitar el razonamiento humano. Así que la lógica difusa en comparación con la lógica clásica permite el trabajo con información que no es estrictamente clara ni exacta para realizar evaluaciones convencionales, en contraste con la lógica clásica que solo trabaja bajo información definida y precisa.

Por otro lado, el paradigma de hardware libre, trae consigo la reutilización y la adaptación de diseños, permitiendo así innovar y mejorar los proyectos de forma colaborativa a nivel mundial, ya que existen comunidades de desarrollo, programación, pruebas, y soporte que día a día crecen de forma dinámica y participativa. Al tener estándares abiertos, disminuye los costos y tiempos de diseños en los proyectos. En el caso de la enseñanza es de gran utilidad, ya que es una herramienta que permite interactuar de una forma dinámica y asequible, ya que cuenta con un amplio desarrollo de la comunidad de hardware libre, en la cual se puede apoyar para afianzar conocimientos y lo levar a cabo una implementación más eficiente, en menor tiempo y con menos recursos.

[1] PALMER, Pol; MORENO, Montaño. ¿Qué son las redes neuronales artificiales? Aplicaciones realizadas en el ámbito de las adicciones [en línea]. Islas Baleares: Universidad de las Islas Baleares, 1999. [Consultado 21/11/2015]. Disponible en internet: http://disi.unal.edu.co/~lctorress/RedNeu/LiRna001.pdf.

1. PLANTEAMIENTO DEL PROBLEMA.

El interés de la enseñanza actual se enfoca a que el estudiante pueda aprender de una forma más práctica, innovadora y creativa. Donde se le ofrecen las herramientas y los bloques estructurales de una técnica en específico, esperando de esto una respuesta del incentivo en la actitud del estudiante, que permita aumentar su curiosidad innata y así elaborar soluciones a problemas cotidianos presentes en su entorno, aumentando así su propia experiencia y conocimiento en los sistemas y lenguajes utilizados. Para el desarrollo de este proyecto, una de las opciones más apropiadas y relevantes para llevar a cabo la implementación y aplicación de estas prácticas, es la utilización de plataformas de hardware libre, permitiendo un reducido gasto económico y una evolución conjunta a lo largo de la globalización del conocimiento académico actual.[2]

La asequibilidad de plataformas hardware libre que existe actualmente en el mundo académico y estudiantil, permite darse cuenta de un potencial que ha sido menospreciado y pobremente utilizado en los ámbitos de la practicidad y en la posibilidad de dar a los estudiantes un acercamiento, por medio de prácticas de laboratorio, a lo que serían los sistemas inteligentes en un campo real de aplicación, como puede ser industrial, médico, aeronáutico, etc.

En base a lo anterior se plantea la siguiente pregunta:

¿Es posible diseñar una serie de herramientas, que generen una experiencia de aprendizaje más interactiva, las cuales ayuden a propiciar una mejor adquisición de conocimiento por parte de los estudiantes en el área de sistemas inteligentes usando hardware libre?

[2] BUITRAGO, Manuel. Capítulo 2: Las redes neuronales artificiales [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2015. [Consultado 21/11/2016]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/navarrete_g_j/capitulo2.pdf.

2. JUSTIFICACIÓN.

Con el desarrollo de mecanismos cada vez más eficientes y complejos, además, de que la educación ha tratado de expandirse buscando la incursión de nuevas formas de hacer llegar el conocimiento a las personas, una de ellas es la creación de herramientas que permitan el acercamiento real de los estudiantes a ciertas situaciones y problemas que en ocasiones no se pueden presentar en forma tangible o física en cuanto a desarrollo de experimentos de laboratorio se refiere. Es importante resaltar que, con el desarrollo de estas herramientas de hardware libre, se puede innovar en el campo de la docencia, ya que se puede ver como un prototipo inicial, para implementarlo en proyectos similares en otras áreas.[3]

Actualmente la Universidad Autónoma de Occidente cuenta en su pensum educativo de Ingeniería Mecatrónica con una materia de sistemas inteligentes, esta asignatura además de que forma parte del pensum de ingeniería Mecatrónica, es electiva para otras ingenierías como biomédica y electrónica la cual hace que esta asignatura se pueda ver desde otras perspectivas y se puedan enfatizar en los conocimientos de ciertos modelos de control, como lo son las redes neuronales y la lógica difusa, en ámbitos industriales, lo que hace que este proyecto pueda ser visto como una herramienta educativa práctica que complemente lo visto en el curso.

El presente proyecto se enfoca en la capacidad de implementación presente en las plataformas de hardware libre con sistemas inteligentes permitiendo cumplir el objetivo principal de crear un incentivo hacia el estudiante acerca de esta área, el cual al ver que los conocimientos adquiridos pueden ser fácilmente aplicados en una herramienta abierta y libre a la comunidad, además por su bajo costo permitiría que el estudiante las pueda replicar, aumentando su nivel de aprendizaje y así podrá enriquecer su propia experiencia, llegando a nuevos niveles de profundización en la técnica de desarrollo de sistemas inteligentes.

Además, se tiene en cuenta que este proyecto será pionero en este tipo de aplicaciones y prácticas de laboratorio orientadas a temas de sistemas inteligentes, haciendo énfasis en que cada práctica tendrá un manual claro y conciso de cómo llevar a cabo su implementación.

[3] LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

3. OBJETIVOS

3.1 OBJETIVO GENERAL.

Implementar herramientas que faciliten la enseñanza de conceptos relacionados con redes neuronales artificiales y lógica difusa, usando plataformas de hardware libre.

3.2 OBJETIVOS ESPECÍFICOS.

- Realizar un estudio de las generalidades de los sistemas inteligentes.
- Seleccionar algunos conceptos de redes neuronales y lógica difusa que puedan ser implementar en plataformas de hardware libre.
- Implementar los conceptos seleccionados con las plataformas de hardware libre más adecuadas.
- Generar las guías correspondientes de cada una de las herramientas implementadas.

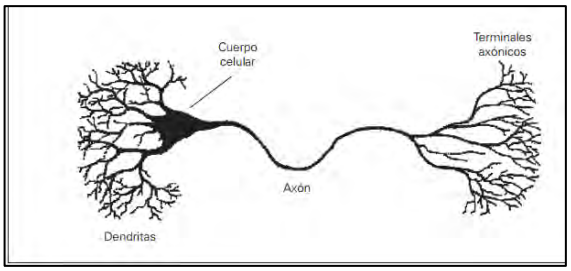
4. MARCO TEORICO

En este capítulo se presentarán los conceptos usados para la elaboración de este proyecto. Se comienza con una breve descripción del funcionamiento de una red neuronal, los tipos de arquitectura (monocapa- multicapa), perceptrón, perceptrón multicapa (MLP), el entrenamiento, validación, test de una red neuronal y su respectivo aprendizaje de funciones, reconocimiento de caracteres y patrones con redes neuronales, Luego se presenta el concepto de lógica y control difuso, con sus referentes aplicaciones, por último, se muestra las diferentes plataformas de hardware libre.

4.1 RED NEURONAL

Las neuronas biológicas (Ilustración 1) se caracterizan por su capacidad de comunicarse. Las dendritas y el cuerpo celular de la neurona reciben señales de entrada excitatorias e inhibitorias de las neuronas vecinas; el cuerpo celular las combina e integra y emite señales de salida. El axón transporta esas señales a los terminales axónicos, que se encargan de distribuir información a un nuevo conjunto de neuronas. Por lo general, una neurona recibe información de miles de otras neuronas y, a su vez, envía información a miles de neuronas más.

Ilustración 1.Estructura Neurona biológica

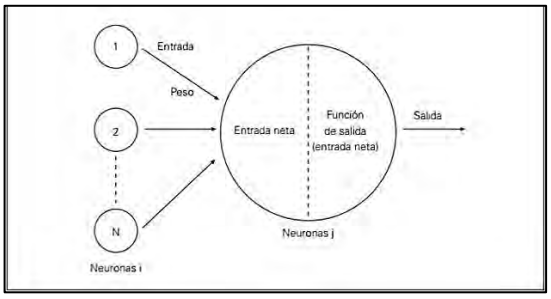


Fuente: PALMER, Pol; MORENO, Montaño. ¿Qué son las redes neuronales artificiales? Aplicaciones realizadas en el ámbito de las adicciones [en línea]. Islas Baleares: Universidad de las Islas Baleares, 1999. [Consultado 21/11/2015]. Disponible en internet: http://disi.unal.edu.co/~lctorress/RedNeu/LiRna001.pdf.

Por su parte, la neurona artificial pretende mimetizar las características más importantes de la neurona biológica. En general, recibe las señales de entrada de las neuronas vecinas ponderadas por los pesos de las conexiones. La suma de estas señales ponderadas proporciona la entrada total o neta de la neurona y, mediante la aplicación de una función matemática — denominada función de

salida—, sobre la entrada neta, se calcula un valor de salida, el cual es enviado a otras neuronas (Ilustración 2). Tanto los valores de entrada a la neurona como su salida pueden ser señales excitatorias (cuando el valor es positivo) o inhibitorias (cuando el valor es negativo).

Ilustración 2.Funcionamiento neurona artificial



Fuente: PALMER, Pol; MORENO, Montaño. ¿Qué son las redes neuronales artificiales? Aplicaciones realizadas en el ámbito de las adicciones [en línea]. Islas Baleares: Universidad de las Islas Baleares, 1999. [Consultado 21/11/2015]. Disponible en internet: http://disi.unal.edu.co/~lctorress/RedNeu/LiRna001.pdf.

4.2 ARQUITECTURA DE REDES NEURONALES ARTIFICIALES (MONOCAPA Y MULTICAPA)

La capacidad de cálculo y potencia de la computación neuronal proviene de las múltiples conexiones de las neuronas artificiales que constituyen las redes ANN, este término es en realidad la definición de una clase de tales funciones (donde los miembros de la clase se obtienen variando parámetros, los pesos de conexión, o específicos de la arquitectura, tales como el número de neuronas o su conectividad).

Un ANN se define típicamente por tres tipos de parámetros:

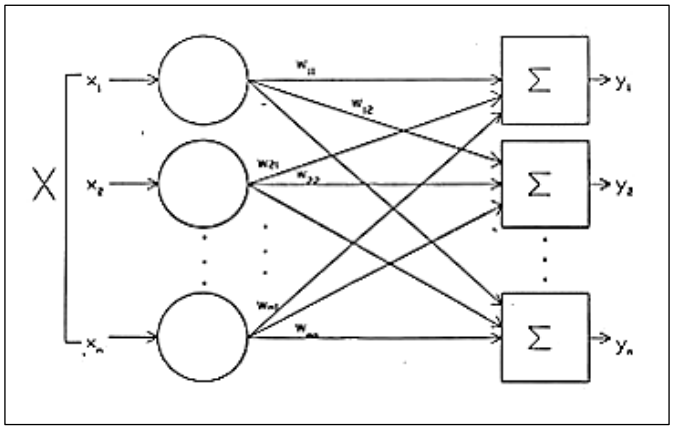
- El patrón de interconexión entre las diferentes capas de neuronas
- El proceso de aprendizaje para la actualización de los pesos de las interconexiones
- La función de activación que convierte la entrada ponderada de una neurona a su activación de la salida.

Las neuronas que componen una RNA se organizan de forma jerárquica formando capas. Una capa o nivel es un conjunto de neuronas cuyas entradas de información provienen de la misma fuente (que puede ser otra capa de neuronas) y cuyas salidas de información se dirigen al mismo destino (que puede ser otra capa de neuronas). En este sentido, se distinguen tres tipos de capas: la capa de entrada recibe la información del exterior; la o las capas ocultas son aquellas cuyas entradas y salidas se encuentran dentro del sistema y, por tanto, no tienen contacto con el exterior; por último, la capa de salida envía la respuesta de la red al exterior.

4.3 RED MONOCAPA

La red más simple es un grupo de neuronas ordenadas en una capa como se muestra en la Ilustración (3). Los nodos circulares sólo son distribuidores de las entradas y no se consideran constituyentes de una capa.

Ilustración 3. Red Neuronal de una Capa.



Fuente: LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

Cada una de las entradas está conectada a través de su peso correspondiente a cada neurona artificial. En la práctica existen conexiones eliminadas e incluso conexiones entre las salidas y entradas de las neuronas de una capa. No obstante, la figura muestra una conectividad total por razones de generalización.

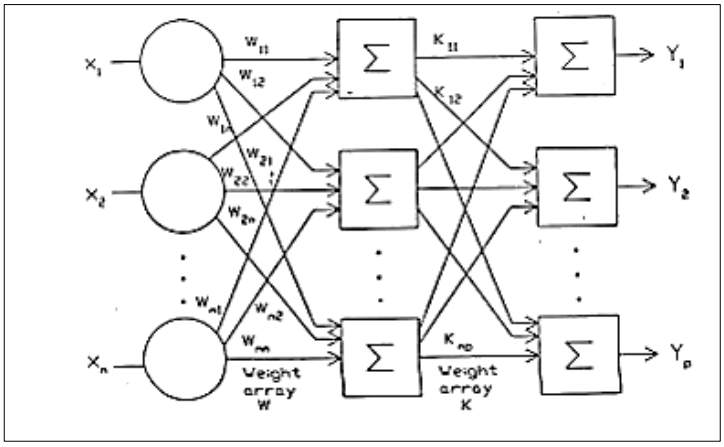
Normalmente las redes más complejas y más grandes ofrecen mejores prestaciones en el cálculo computacional que las redes simples. Las configuraciones de las redes construidas presentan aspectos muy diferentes, pero tienen un aspecto común, el ordenamiento de las neuronas en capas o niveles imitando la estructura de capas que presenta el cerebro en algunas partes.

4.4 REDES MULTICAPA

Las redes multicapa disponen de conjuntos de neuronas agrupadas en dos o más capas. En la mayoría de casos, este tipo de redes están formadas por una capa de entrada, una capa de salida y una o más capas intermedias u ocultas; donde la información se transmite desde la capa de entrada hasta la capa de salida y donde cada neurona está conectada con todas las neuronas de la siguiente capa (en la ilustración 4 se muestra un ejemplo de red multicapa).

Las redes multicapa se forman con un grupo de capas simples en cascada. La salida de una capa es la entrada de la siguiente capa. Se ha demostrado que las redes multicapa presentan cualidades y aspectos por encima de las redes de una capa simple. La Ilustración (4) muestra una red de dos capas.

Ilustración 4. Red Neuronal de dos Capas.



Fuente: LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

Conviene destacar que la mejora de las redes multicapa estriba en la función de activación no lineal entre capas, pudiéndose llegar al caso de diseñar una red de una capa simple equivalente a una red multicapa si no se utiliza la función no lineal de activación entre capas. [4]

4.5 FUNCIÓN DE ACTIVACIÓN TANSIG CAPAS OCULTAS

La función de transferencia tangente hiperbólica (tansig) es una de las funciones más utilizadas en las redes neuronales por su flexibilidad y el amplio rango de resultados que ofrece. La función tansig permite a las redes aprender variaciones no lineales de distintos tipos de ambientes es decir que da resultados entre 1 y -1, por lo que se amplía a los números negativos los posibles resultados (acota la respuesta).

La ecuación de la salida de en nuestro caso de una función tansig es igual a:

$$Salida = \frac{2}{(1 + e^{-2n}) - 1}$$

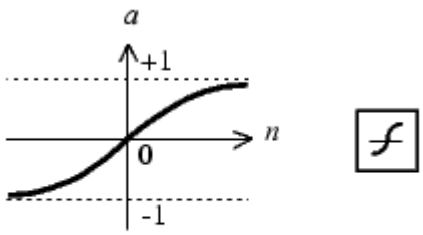
Donde n=Neta. La Neta es igual a:

$$Neta = \left(\sum_{j=0}^{j} W_{jk} P_j \right) + b_k$$

la representación gráfica de una función tansig es la siguiente:

[4] MEJÍA, S. Perceptrón Multicapa [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2012 [Consultado 12/03/2017]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf.

Ilustración 5. Grafica función tansig



$a = tansig(n)$

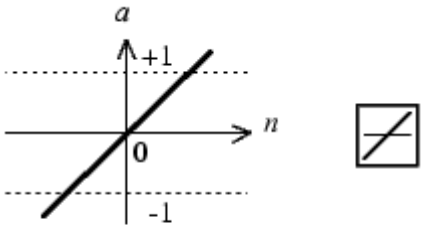
Tan-Sigmoid Transfer Function

Fuente: LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

4.6 FUNCIÓN DE ACTIVACIÓN LINEAL (PURELIN) CAPA DE SALIDA

La salida de una función de transferencia lineal es igual a su entrada, es decir, Salida=Neta. Y su representación gráfica está dada por la Ilustración 6.

Ilustración 6. Función Purelin.

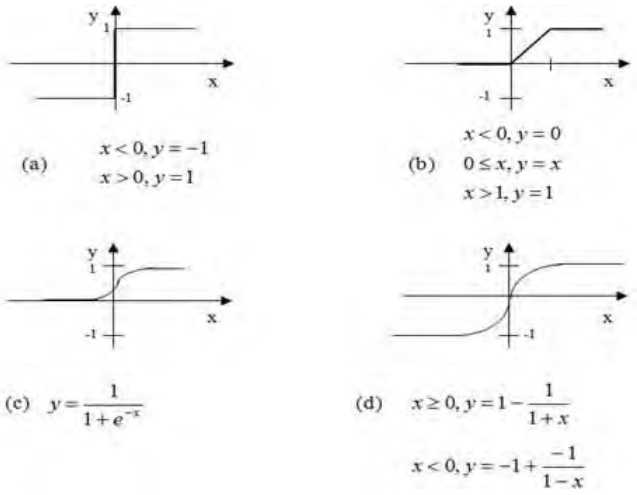


$a = purelin(n)$

Linear Transfer Function

Fuente: LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

Ilustración 7. Otras funciones de redes neuronales



Fuente: LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

4.7 PERCEPTRÓN SIMPLE

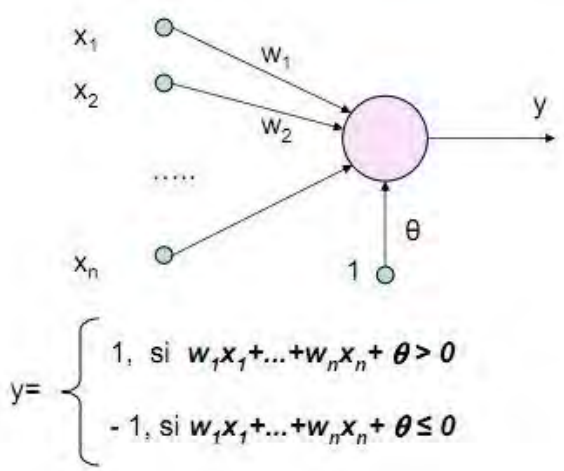
El Perceptrón Simple, es el primer modelo de red neuronal artificial, desarrollado por Rosenblatt, 1959. El perceptrón es un modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entrada y otra de salida. La operación de una red de este tipo con ***"n"*** neuronas de entrada y otra de salida con ***m*** neuronas. Utiliza señales binarias, tanto de entrada como de salida de las neuronas y su modo de entrenamiento es supervisado.

Como regla de propagación se usa la suma ponderada del producto escalar por el vector de entrada, debiendo superarse un cierto umbral Θ_i:

$$y_i(t) = \begin{cases} 1 & \text{si } \Sigma w_{ij} x_j > \Theta_i \\ 0 & \text{si } \Sigma w_{ij} x_j <= \Theta_i \end{cases}$$

El perceptrón simple sólo sirve para clasificar problemas linealmente separables, cosa que ya se podía hacer mediante métodos estadísticos, y de una forma mucho más eficiente. Y su arquitectura se muestra en la Ilustración 8.[5]

Ilustración 8. Arquitectura de un Perceptrón simple.



Fuente: MEJÍA, S. Perceptrón Multicapa [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2012 [Consultado 12/03/2017]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf.

4.8 PERCEPTRÓN MULTICAPA

El Perceptrón Multicapa (MLP), debido al grupo PDP (*paralell distributed processing*) de 1986, es una extensión y generalización del perceptrón simple en el que:

· Se añaden una o más capas ocultas. A veces una capa oculta es suficiente.

· Se permiten entradas continuas.

· Las funciones de activación son de tipo sigmoide (puede ser lineal en la capa de salida).

[5] Ibid., p. 21.

Su principal ventaja e innovación es su algoritmo de aprendizaje, el algoritmo de retropropagación (en inglés *backpropagation*, BP), debido a Werbos, Rumelhart y Parker.

Estas ecuaciones forman la llamada regla delta generalizada.

La regla delta generalizada es en el fondo un algoritmo del gradiente, que trata de minimizar el error cuadrático medio (MSE – *medium square* e*rror*):

- Se aplica una pareja de ejemplo (x,y)
- Se propaga hacia adelante y calculamos la salida real y'
- Se calcula el error según la diferencia entre y e y'
- Se van modificando los pesos hacia atrás
- Volvemos al paso 1 hasta que se alcance un cierto valor mínimo de error.

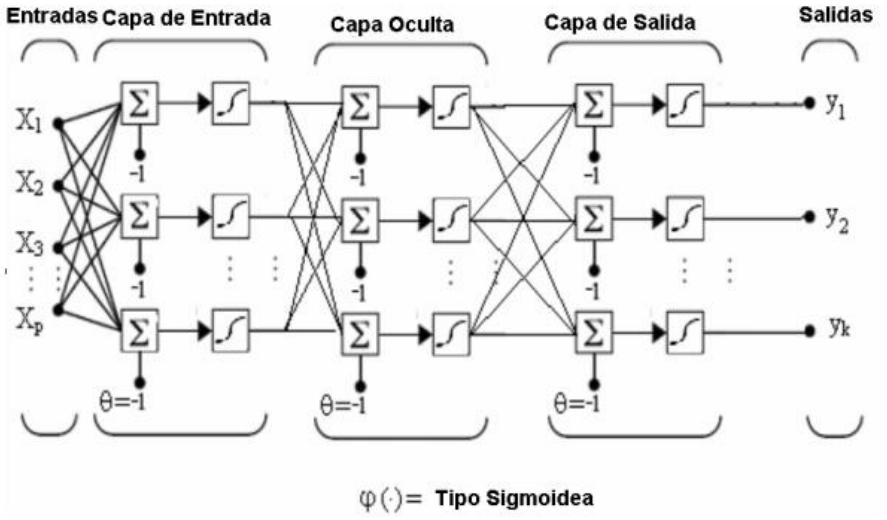
La arquitectura de Perceptrón multicapa se caracteriza porque tiene sus neuronas agrupadas en capas de diferentes niveles. Cada una de las capas está formada por un conjunto de neuronas y se distinguen tres tipos de capas diferentes: la capa de entrada, las capas ocultas y la capa de salida.

Las neuronas de la capa de entrada no actúan como neuronas propiamente dichas, sino que se encargan únicamente de recibir las señales o patrones del exterior y propagar dichas señales a todas las neuronas de la siguiente capa. La última capa actúa como salida de la red, proporcionando al exterior la respuesta de la red para cada uno de los patrones de entrada. Las neuronas de las capas ocultas realizan un procesamiento no lineal de los patrones recibidos.

Como se observa en la Ilustración 9, las conexiones del Perceptrón multicapa siempre están dirigidas hacia adelante, es decir, las neuronas de una capa se conectan con las neuronas de la siguiente capa, de ahí que reciban también el nombre de redes alimentadas hacia adelante o redes "*feedforward*". Las conexiones entre las neuronas de la red llevan también asociado un umbral, que el caso del Perceptrón multicapa suele tratarse como una conexión más a la neurona, cuya entrada es constante e igual a 1. La función de activación es una

función que trabaja en zonas no lineales, es importante que esta función sea diferenciable para poder aplicar el algoritmo de propagación hacia atrás. [6]

Ilustración 9. Arquitectura del perceptrón multicapa



Fuente: RUEDA, Albert. Capítulo 3 perceptrón multicapa [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2014 [Consultado 10/03/2107]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf

4.9 ENTRENAMIENTO DE LAS REDES NEURONALES ARTIFICIALES

Una de las principales características de las ANN es su capacidad de aprendizaje. El entrenamiento de las ANN muestra algunos paralelismos con el desarrollo intelectual de los seres humanos. No obstante, aun cuando parece que se ha conseguido entender el proceso de aprendizaje conviene ser moderado porque el aprendizaje de las ANN está limitado.

El objetivo del entrenamiento de una ANN es conseguir que una aplicación determinada, para un conjunto de entradas produzca el conjunto de salidas deseadas o mínimamente consistentes. El proceso de entrenamiento consiste en la aplicación secuencial de diferentes conjuntos o vectores de entrada para que se

[6] RUEDA, Albert. Capítulo 3 perceptrón multicapa [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2014 [Consultado 10/03/2107]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf

ajusten los pesos de las interconexiones según un procedimiento predeterminado. Durante la sesión de entrenamiento los pesos convergen gradualmente hacia los valores que hacen que cada entrada produzca el vector de salida deseado.

Dada una estructura y tamaño de la red neuronal, se procede al entrenamiento de la red. El entrenamiento o aprendizaje, cuyo objetivo es que la red neuronal sea capaz de reproducir el comportamiento subyacente en los datos aportados, consiste básicamente en la minimización de una función de coste o error, lo que equivale a que la salida de la red, se aproxima a la salida en los datos. La función de coste más común es la de promedio de errores al cuadrado (MSE).

Para la optimización de la red neuronal, se emplean diferentes métodos de ajuste de parámetros de la red (pesos de las conexiones y sesgo de las neuronas), a partir de unos valores o bien aleatorios, o bien predefinido (inicialización de la red). Algunos ejemplos de los métodos de ajuste son los de tipo gradiente o los algoritmos genéticos:

- Los métodos de tipo gradiente calculan la variación del error al variar cada uno de los parámetros (a modo de derivada multidimensional), y luego modifican todos los parámetros de la red neuronal obteniendo un error menor. Se puede decir que es una búsqueda en serie de la solución o mínimo global.
- Los métodos basados en algoritmos genéticos, consisten en la generación de un determinado número de redes o hijos a través de mutaciones en los parámetros, evaluando el error de la red para cada uno de ellos. Los hijos con menor error, tienen mayor probabilidad de convertirse en padres de nuevas redes, mientras que los hijos con mayor error desaparecen. Se trata de una búsqueda en paralelo de la solución.

Ambos métodos son métodos iterativos, que se repiten hasta cumplir alguno de los diferentes criterios de parada. Algunos ejemplos de los criterios de parada son el número de iteraciones, la obtención de un error mínimo, o un tiempo de ejecución. En cualquier caso, generalmente es difícil asegurar que la solución obtenida no es un mínimo local. [7]

[7] VIVAS, Hevert. Optimización en el entrenamiento del perceptrón multicapa [en línea]. Popayan: Universidad del cauca, 2014. [Consultado 10/03/2017] Disponible en internet: http://www.unicauca.edu.co/matematicas/investigacion/gedi/optimizacion/Vivas.pdf

4.10 DATOS DE ENTRENAMIENTO, VALIDACIÓN Y TEST

Para controlar si una red neuronal ha sobre-aprendido, se dividen los datos en diferentes grupos:

- Datos de entrenamiento. Son los datos empleados en el ajuste de los parámetros de la red neuronal. Han de ser representativos del total de datos, por lo que normalmente se seleccionan aleatoriamente.
- Datos de validación. Se emplean después de cada iteración en el proceso de entrenamiento, para comprobar si se produce el sobre-aprendizaje.
- Datos de test. Sólo se emplean una vez finalizado el entrenamiento.

La división de los datos es normalmente un 80% de datos de entrenamiento, un 10% de validación y un 10% de test, aunque la elección de dichos porcentajes depende del número de datos disponible y de su distribución. Dicha división se puede realizar con algún criterio de modo que los datos de cada grupo sean representativos, o de modo aleatorio.

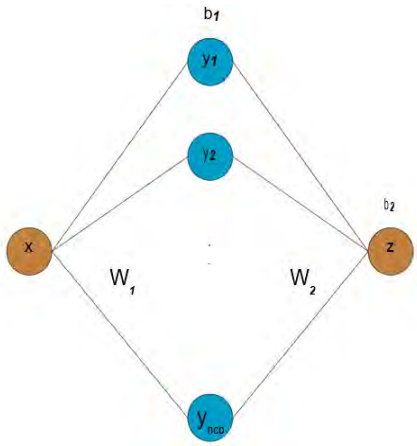
4.11 APRENDIZAJE DE FUNCIONES

Para realizar un aprendizaje de funciones, en este caso de la función seno, se hace uso del perceptrón multicapa como aproximador universal de funciones. En efecto, para cualquier función de R n en R m siempre es posible diseñar y entrenar un perceptrón multicapa, de tal manera que realice un ajuste de los datos de dicha función con un grado de precisión predefinido.

En particular, es relativamente sencillo evaluar una función de variable y valor real tal como la función seno, mediante una de estas redes. Este problema lo resolvimos mediante una perceptrón multicapa de tres capas (Ilustración 10), donde el número de neuronas en la capa oculta (*nco*) es definido por el usuario. Los patrones de entrenamiento fueron $x = (x_1; x_2; : : : ; x_p)^T$; $p = 41$ y $t = (t_1; t_2; : : : ; t_p)^T$ con $t_i = \text{sen } x_i$; donde las componentes del vector x (entradas de la red) son números reales distribuidos uniformemente en el intervalo [0; 2]: Usamos como función de activación la función sigmoidal (logsig) y la identidad (purelin) en la capa oculta y de salida, respectivamente.[8]

[8] Ibid., p. 28.

Ilustración 10. Perceptrón multicapa que aproxima la función seno.



Fuente: VIVAS, Hevert. Optimización en el entrenamiento del perceptrón multicapa [en línea]. Popayan: Universidad del cauca, 2014. [Consultado 10/03/2017] Disponible en internet: http://www.unicauca.edu.co/matematicas/investigacion/gedi/optimizacion/Vivas.pdf

4.12 RECONOCIMIENTO DE CARACTERES

El reconocimiento de caracteres es otra área en la cual las redes neuronales están suministrando soluciones. Algunas de estas soluciones van más allá de simples curiosidades académicas. HNC Inc.., comercializa una red neuronal que puede reconocer caracteres impresos a manos a través de un escáner. Este producto puede tomar tarjetas, como de crédito débito y colocar aquellas que reconoce en una base de datos. Este producto ha estado en el mercado por más de dos años y tiene un 98% de exactitud para los números, un poco menos para los caracteres alfabéticos. Actualmente, el sistema es construido para resaltar caracteres por debajo de un cierto porcentaje de probabilidad de estar bien así que el usuario puede llenar lo que el computador no. Este producto los está usando bancos, instituciones financieras y compañías de tarjetas de crédito.

Las investigaciones más grandes en el campo de reconocimiento de caracteres apuntan a escasear los caracteres orientales en un computador. Hoy en día, estos caracteres requieren 4 o 5 caracteres de computador para cada uno.

4.13 RECONOCIMIENTO DE PATRONES

El reconocimiento de patrones tiene un gran número de aplicaciones. Por ejemplo, detección de bombas en equipajes en los aeropuertos, desde pequeñas variaciones, patrones desde dentro de salida de sensores especializados. Otra aplicación se refiere a como se ha entrenado una red neuronal de propagación hacia atrás que recolecta en salas de emergencia de personas que sienten que están experimentando un ataque cardiaco para suministrar una probabilidad de un ataque real versus una falsa alarma.

No obstante, el uso más importante de las redes neuronales como reconocimiento de patrones está dentro del campo conocido como control de calidad. Un gran número de aplicaciones para calidad automatizada están en uso. Estas aplicaciones están diseñadas para encontrar que una parte en cien o una parte en mil es defectuoso. Los inspectores humanos se tornan fatigados o se distraen. Los sistemas ahora evalúan empalmes de soldadura, cortes y pegados. Un fabricante de autos tiene un sistema prototipo que evalúa el color de la pintura. Este sistema digitaliza gráficos de nuevos lotes de pinturas para determinar si tienen el matiz correcto.

Otra área donde las redes neuronales están siendo construida para el reconocimiento de patrones es en procesadores para sensores. Muchos de estos sensores existen dentro de la industria de defensa. Estos sensores toman datos de cámaras, sistemas sonares, grabadoras sísmicas y sensores infrarrojos. Estos sensores son usados para reconocimiento de fenómenos.

4.14 LÓGICA DIFUSA.

Es una de las tantas ramas en las que el Control Inteligente ha terminado. Y se trata de un control basado en reglas que utiliza técnicas para manejar la imprecisión. Cabría separar el estudio de los controladores difusos como alternativa al control adaptativo, predictivo u otros del control experto que utiliza incertidumbre.

Por lo tal es una de las disciplinas matemáticas que cuenta con el mayor número de seguidores actualmente, la cual utiliza expresiones que no son ni totalmente ciertas ni completamente falsas, es decir, es la lógica aplicada a conceptos que pueden tomar un valor cualquiera de veracidad dentro de un conjunto de valores que oscilan entre dos extremos, la verdad absoluta y la falsedad total. Conviene destacar que lo que es difuso, borroso, impreciso o vago no es la lógica en sí, sino

el objeto que estudia: expresa la falta de definición del concepto al que se le aplica. La lógica difusa permite tratar información imprecisa, como estatura media o temperatura baja, en términos de conjuntos borrosos que se combinan en reglas para definir acciones: si la temperatura es alta entonces enfriar mucho. De esta manera, los sistemas de control basados en lógica difusa combinan variables de entrada, definidas en términos de conjuntos difusos, por medio de grupos de reglas que producen uno o varios valores de salida. [9]

4.15 CONTROL DIFUSO

El control difuso representa actualmente una novedosa e importante rama de la técnica de regulación. Los procedimientos convencionales no se sustituyen, sino que se complementan de forma considerable en función del campo de aplicación.

Los mayores éxitos en el campo de las aplicaciones industriales y comerciales de los métodos difusos los ha logrado hasta la fecha el regulador difuso.

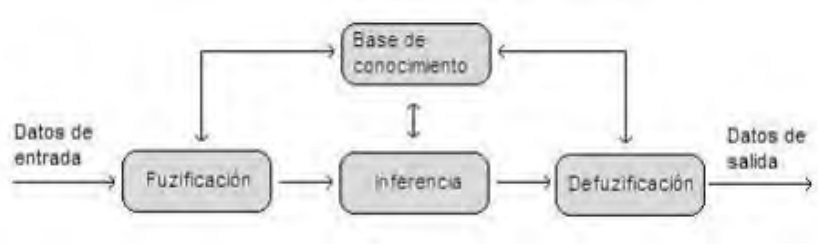
Por esto, al desarrollar un controlador difuso es posible prescindir de la rigidez matemática y transmitir el raciocinio humano hacia un sistema. [10]

La estructura normalmente utilizada en un controlador difuso se puede apreciar en la siguiente Ilustración 11.

[9] HERNÁNDEZ, Alberto. Capítulo 4. Sección del Control Difuso. [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2012 [Consultado 24/11/2015]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/hernandez_b_ii/capitulo4.pdf

[10] CÓRDOBA, Oscar. Métodos de Inferencia [en línea]. Prezi, 2011 [Consultado 24/11/2015]. Disponible en internet: https://prezi.com/5jju05ocq5xr/metodos-de-inferencia/

Ilustración 11. Estructura de un control difuso.



Fuente: CÓRDOBA, Oscar. Métodos de Inferencia [en línea]. Prezi, 2011 [Consultado 24/11/2015]. Disponible en internet: https://prezi.com/5jju05ocq5xr/metodos-de-inferencia/

El primer bloque llamado fuzificador o fuzificación es donde los datos de entrada son procesados para calcular el grado de membresía que tendrán dentro del controlador. Posteriormente se tiene el dispositivo de inferencia que junto con la base de conocimientos realizan la toma de decisiones que dirán la forma en que el sistema actuará. El método de inferencia se basa en el grado de pertenencia de los datos de entrada en los conjuntos difusos de los espacios correspondientes, siempre que éstas se den, para tomar una decisión en el espacio de salida. [11]

Los reguladores difusos son reguladores no-lineales. Por medio de la selección adecuada de funciones de pertenencia y del establecimiento de una base de reglas se pueden compensar no-linealidades en el sistema de regulación de procesos.

Las funciones de pertenencia son modelos matemáticos para los términos lingüísticos, como p. ej. las funciones de pertenencia triangulares, trapezoidales o gaussianas.

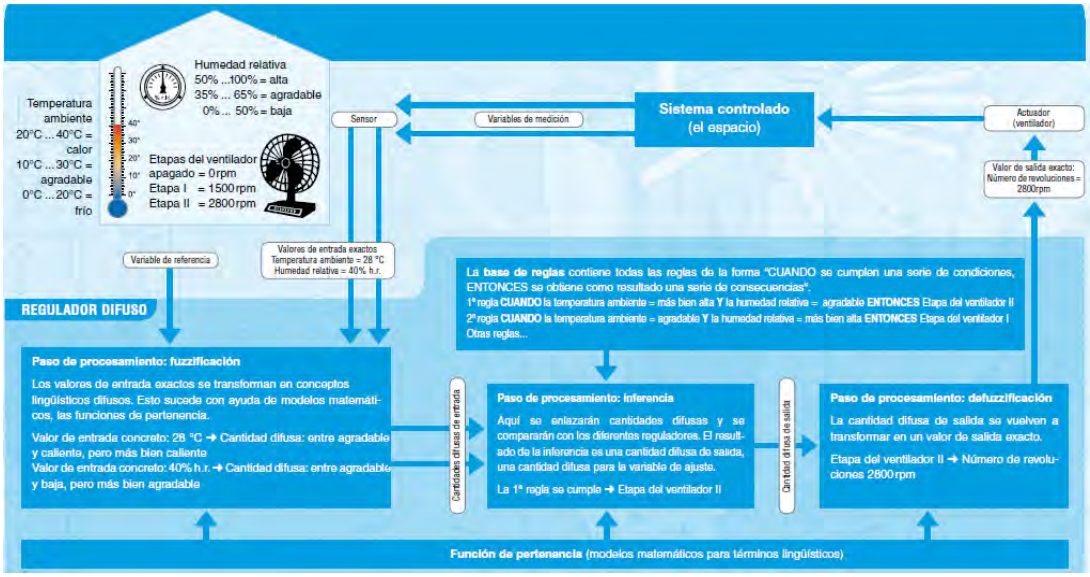
Como en el caso de un regulador convencional, en el regulador difuso se transforman variables de entrada en variables de salida, que actúan en el proceso o en el sistema de control. Múltiples variables de entrada y salida se pueden enlazar entre sí de forma que sistemas complejos se pueden regular fácilmente. Los valores de entrada y salida son valores exactos en forma de señales.

[11] Ibid., p. 31.

La imprecisión típica de los métodos difusos desempeña un papel sólo dentro del regulador. En un regulador difuso se ejecutan tres pasos de procesamiento: fuzificación, inferencia y defuzzificación. El diseño de un regulador difuso contiene la selección de variables de entrada/ salida, el establecimiento de funciones de pertenencia y la disposición de la base de reglas.

En este sencillo ejemplo como se muestra en la ilustración 12 se debe influenciar el clima ambiental con un ventilador. Dependiendo de la temperatura ambiente y de la humedad relativa funcionará un ventilador con diferentes etapas. El modelo descriptivo muestra las secuencias en un regulador difuso, sin pretensiones de una verdadera realización.

Ilustración 12. Ejemplo de un control difuso (Temperatura y humedad relativa), con ventilador.



Fuente: GUNT, Hamburg. Control Difuso [en línea]. 2015 [Consultado 10/03/2017]. Disponible en internet: http://www.gunt.de/images/download/Conocimientos-bsicos-control-difuso_spanish.pdf

Es claro pensar que la aplicación de la lógica difusa directa será en los sistemas de control difuso, los cuales utilizan términos difusos para así formular reglas de control para el sistema. Como la lógica difusa sugiere un cierto grado de pertenencia para un dato que se presente dentro de los conjuntos difusos, permite a un controlador difuso tomar diferentes grados de acción en un sistema.

El control difuso puede aplicarse en innumerables sistemas, tanto sencillos, como brazos articulados y vehículos autónomos, en los cuales los modelos matemáticos son muy complejos; así que empleando técnicas de razonamiento aproximado es posible controlar sistemas complejos cuando el entorno no se conoce de forma precisa. Dicha característica permite mayor flexibilidad que el control clásico en el que para la realización de un controlador se requiere de un alto grado de cálculo matemático.

4.16 CONJUNTOS DIFUSOS.

Conjunto difuso. Es un conjunto sin un límite definido. La transición entre pertenecer a un conjunto" y "no pertenecer a un conjunto" es gradual y esta transición suave es caracterizada por una función de pertenencia. Los conjuntos definidos de forma imprecisa desempeñan un papel importante en el pensamiento humano, particularmente en los dominios del reconocimiento de patrones, de la comunicación de la información y de la abstracción.

4.16.1 Razonamiento aproximado. El razonamiento con lógica difusa no es exactosino en cierta forma impreciso. De acuerdo con las premisas y las implicaciones difusas las conclusiones que se obtienen o se deducen igualmente difusas.

4.16.2 Variable lingüística. Este concepto fue introducido para proporcionar una base para el razonamiento aproximado, así: "por una variable lingüística se quiere decir que se trata de una variable cuyos valores son palabras u oraciones en un lenguaje natural o artificial. La motivación para el uso de palabras o de oraciones en lugar de números es que las caracterizaciones lingüísticas son, en general, menos precisas que los valores numéricos". La matemática de los conjuntos difusos trabaja con conjuntos que no tienen límites perfectamente definidos, es decir, la transición entre la pertenencia y no-pertenencia de una variable a un conjunto es gradual. Estos conjuntos se caracterizan por las funciones de pertenencia, que dan flexibilidad a la modelación utilizando expresiones lingüísticas, tales como mucho, poco, leve, severo, escaso, suficiente, caliente, frío, joven, viejo, etc. Surgió de la necesidad de solucionar problemas complejos con información imprecisa, para los cuales la matemática y lógica tradicionales no son suficientes.

La lógica difusa es un lenguaje que permite trasladar sentencias sofisticadas del lenguaje natural a un formalismo matemático.

4.16.3 Características generales de los conjuntos difusos. Estas son algunas de las características matemáticas de los conjuntos difusos. En la ilustración 13 se presentan algunas de las características de estos conjuntos.

Conjunto difuso: Expresa el grado de pertenencia al conjunto que tiene cada uno de los elementos. El conjunto difuso A en X puede definirse como el conjunto definido.

$$A = \{(x, \mu_A(x)) | x \in X\}$$

donde $\mu_A(x)$ es la función de pertenencia al conjunto difuso.[12]

4.16.3.1 Funciones de pertenencia. Dan para cada elemento de X un grado de membresía al conjunto A. El valor de esta función está en el intervalo entre 0 y 1, siendo 1 el valor para máxima pertenencia. Si el valor de esta función se restringiera solamente a 0 y 1, se tendría un conjunto clásico, o no-difuso. Esta función no es única. Las funciones utilizadas más frecuentemente son las de tipo trapezoidal, singleton, triangular (T), tipo S, exponencial, tipo Π (forma de campana).

4.16.3.2 Apoyo. En un conjunto difuso A es el conjunto de todos los puntos x para los cuales la función de pertenencia ($\mu_A(x)$) es mayor que cero.

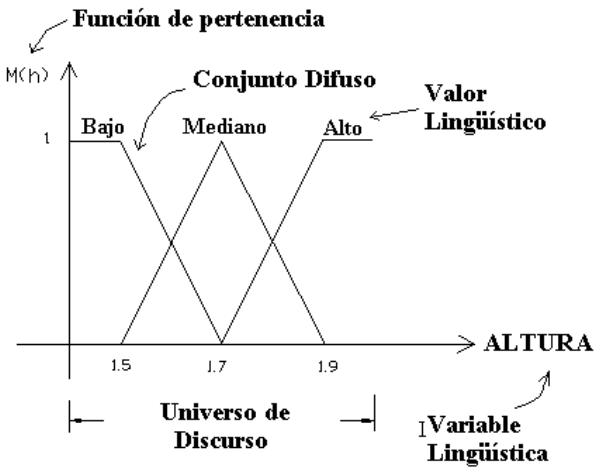
4.16.3.3 Centro. En un conjunto difuso A es el conjunto de todos los puntos para los cuales la función de pertenencia ($\mu_A(x)$) es igual a 1.

4.16.3.4 Normalidad. Un conjunto difuso es normal si siempre existe un punto para el cual la función de pertenencia es 1, es decir el centro no está vacío.

4.16.3.5 Conjunto difuso simple (Singleton). Es el conjunto difuso para el cual el apoyo es solamente un punto, en el cual el valor de la función de pertenencia es 1. Conjunto α-corte: Aα de un conjunto difuso A todos los puntos x para los que se cumple $A_\alpha = \{x \mu_A(x) \geq \alpha\}$

[12] GIRÓN, Jeison. Capítulo 1. Teoría de Conjuntos Difuso. [en línea]. Barcelona: Universidad de la Salle, 2009 [Consultado 24/11/2016]. Disponible en internet: http://users.salleurl.edu/~se04184/P2Definitions/10803/6241/12Mct12de15.pdf;jsessionid=3D12C80C77886D37F62BF5F3B19989BA?sequence=12.

Ilustración 13. Algunas características de un conjunto difuso.



Fuente: VELÁSQUEZ, John. Capítulo 4. Sección del Conjuntos Difuso. [en línea]. Barcelona: Universidad de la Salle, 2010 [Consultado 24/11/2015]. Disponible en internet: http://users.salleurl.edu/~se04184/P2Definitions.html

4.16.3.6 Números difusos. Es un conjunto difuso en la recta real (R) que satisface las condiciones de normalidad y convexidad, la función de pertenencia es continua a trozos y el centro del centro del conjunto difuso consiste de un único valor.

4.16.3.7 Intervalo difuso. Un conjunto difuso es un intervalo difuso si en la recta real (R) que satisface las condiciones de normalidad y convexidad y la función de pertenencia es continua a trozos.

4.16.3.8 Simetría. Un conjunto difuso es simétrico si alrededor de un punto x=c se cumple $\mu_A (c + x) = \mu_A (c - x)$, para todo $x \in X$.

4.16.3.9 Conjunto difuso abierto a la derecha. Un conjunto difuso A es abierto a la derecha si cumple que $\lim_{x \to -\infty} \mu_A (x) = 1$ y $\lim_{x \to +\infty} \mu_A (x) = 0$.

4.16.4 Razonamiento Difuso. También llamado razonamiento aproximado es un

procedimiento de inferencia que saca conclusiones con reglas *si-entonces* utilizando conjuntos difusos Siendo *A, A',* y *B* conjuntos difusos que pertenecen a *X, X,* y *Y* respectivamente. Asume que la implicación difusa $A \rightarrow B$ es expresada como una relación R en $X \times Y$. El conjunto difuso *B* inducido por *"x* es *A"* y la regla difusa *"*si *x* es *A* entonces *y* es *B"*

De acuerdo al problema que se desea resolver se han desarrollado diferentes reglas para el razonamiento difuso, en estas puede variar el número de antecedentes y de consecuencias. Se emplea también diferentes operadores para relacionar los conjuntos difusos como Y, O, Implicación, Defuzzificación, entre otros. [13]

4.17 APLICACIONES DE LÓGICA DIFUSA

La lógica difusa se utiliza cuando la complejidad del proceso en cuestión es muy alta y no existen modelos matemáticos precisos, para procesos altamente no lineales y cuando se envuelven definiciones y conocimiento no estrictamente definido (impreciso o subjetivo).

La teoría de conjuntos difusos ha sido ampliamente aplicada en campos como: la Medicina, Economía, Ecología y Biología. Se ha empleado en empresas de producción de artículos eléctricos y electrónicos como una herramienta de control, se ha utilizado para el desarrollo de procesadores y computadoras.

Los conjuntos difusos son usados para toma de decisiones y estimaciones en Sistemas de Control como son: aire acondicionado, control de automóviles y controladores en sistemas industriales.

En cambio, no es una buena idea usarla cuando algún modelo matemático ya soluciona eficientemente el problema, cuando los problemas son lineales o cuando no tienen solución. Esta técnica se ha empleado con bastante éxito en la industria, principalmente en Japón, extendiéndose sus aplicaciones a multitud de campos. La primera vez que se usó de forma importante fue en el metro japonés, con excelentes resultados. Posteriormente se generalizó según la teoría de la incertidumbre desarrollada por el matemático y economista español Jaume Gil Aluja.

En el campo de la Ingeniería Civil, la lógica difusa está siendo aplicada también en control de cierre de compuertas en presas (Chile), control de tráfico (Puerto Rico), control de secaderos de hoja de tabaco (Cuba), control de balanceo en puentes grúa control de nivel de líquidos en contenedores y se espera que su aplicación se generalizará de manera muy notable en los próximos años.

[13] VELÁSQUEZ, John. Capítulo 4. Sección del Conjuntos Difuso. [en línea]. Barcelona: Universidad de la Salle, 2010 [Consultado 24/11/2015]. Disponible en internet: http://users.salleurl.edu/~se04184/P2Definitions.html

A continuación, se citan algunos ejemplos de su aplicación:

- Sistemas de control de acondicionadores de aire
- Sistemas de foco automático en cámaras fotográficas
- Electrodomésticos familiares (frigoríficos, lavadoras...)
- Optimización de sistemas de control industriales
- Sistemas de escritura
- Mejora en la eficiencia del uso de combustible en motores
- Sistemas expertos del conocimiento (simular el comportamiento de un experto humano)
- Tecnología informática
- Bases de datos difusas: Almacenar y consultar información imprecisa. Para este punto, por ejemplo, existe el lenguaje FSQL.
- En general, en la gran mayoría de los sistemas de control que no dependen de un Sí/No.

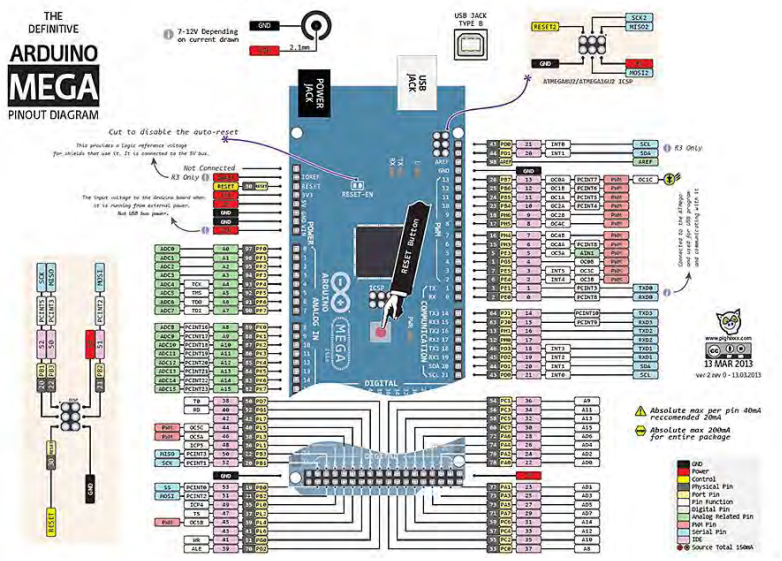
En el campo de la Ingeniería Civil, la lógica difusa está siendo aplicada también en control de cierre de compuertas en presas (Chile), control de tráfico (Puerto Rico), control de secaderos de hoja de tabaco (Cuba), control de balanceo en puentes grúa control de nivel de líquidos en contenedores y se espera que su aplicación se generalizará de manera muy notable en los próximos años.

4.18 PLATAFORMAS DE HARDWARE LIBRE

4.18.1 Arduino. Arduino es una plataforma libre de computación de bajo coste basada en una placa de entrada-salida y en un entorno de desarrollo IDE que implementa el lenguaje *processing/wiringhardware.* Arduino se puede usar para desarrollar objetos interactivos automáticos o conectarse a software en el ordenador (*pure data, flash*, *processing;* MaxMSP.)[14]

[14] BURQUILLOS, Alexis. El microcontrolador Arduino [en línea]. [Consultado 24/11/2015]. Disponible en internet: https://tecnopujol.files.wordpress.com/2009/12/arduino.pdf

Ilustración 14. Estructura Arduino



Fuente: BIGTRONICA, Soluciones electrónicas [en línea]. Medellin,2004 [Consultado 10/03/2017]. Disponible en internet: http://bigtronica.com/tarjetas/11-arduino-mega-2560-5053212000110.html

4.18.2 Raspberry Pi. Ordenador del tamaño de una tarjeta de crédito que consta de una placa base sobre la que se ensambla un procesador, un chip gráfico y memoria RAM.

Fue lanzado en 2009 por la Fundación Raspberry Pi para estimular la enseñanza de informática en escuelas de todo el mundo. Esta propuesta cuenta con una notable comunidad de desarrolladores dispuestos a compartir pasó a paso las nuevas características que van encontrándole a la placa.

Ilustración 15. Estructura de un Raspberry Pi.

Fuente: PE, Isaac. Comparativa y análisis: Raspberry Pi vs competencia [en línea]. [Consultado 24/11/2015]. Disponible en internet: http://comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/

• **BeagleBoard.** Es una placa *open-source* como la Raspi que está producida por Texas Instruments en asociación con Digi-Key. Por ello su cerebro es un SoC OMAP3530 basado en ARM Cortex A8, con un DSP TMS320C64x y una GPU PowerVR SGX530 de *imagination technologies*. Tiene ranura SD/MMC, USB, RS-232, jacks, etc. En la placa también se incluye memoria RAM de 256MB y un flash de 256 MB. Posteriormente surgieron otros modelos derivados con características modificadas denominados BeagleBone, BeagleBoard-xM y BeagleBone Black.[15]

Ilustración 16. Estructura de un BeagleBoard.

Fuente: PE, Isaac. Comparativa y análisis: Raspberry Pi vs competencia [en línea]. [Consultado 24/11/2015]. Disponible en internet: http://comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/

[15] Ibid., p. 39.

4.18.3 CubieBoard. Con sus modelos 1, 2 y 3, la CubieBoard es otra interesante placa que compite con la Raspi. Creada en Shenzhen, es un prototipo presentado en octubre de 2012 capaz de correr Android 4 ICS y Ubuntu, así como otras distros como Fedora, XBMC (ya saben, ahora llamado Kodi), Arch Linux, Debian, etc. Usa un SoC AllWinner A10 (sustituida en su última versión por un A20) con CPU ARM Cortex A8 a 1Ghz, GPU Mali 400MP, acelerador de video CedarX, *display* controller, 1GB de DDR3, 4GB de NAND flash ampliable por microSD y SATA, puerto Ethernet, USB, I2C, SPI y LVDS. Este mismo año también ha aparecido el modelo Cubieboard 8, un modelo mucho más potente que los anteriores, con un A80 basado en ARM Cortex-A15 de cuatro núcleos y otros cuatro núcleos ARM Cortex-A7, GPU PowerVR G6230 y soporte para USB 3.0.[16]

Ilustración 17. Estructura de una CubieBoard.

Fuente: PE, Isaac. Comparativa y análisis: Raspberry Pi vs competencia [en línea]. [Consultado 24/11/2015]. Disponible en internet: http://comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/

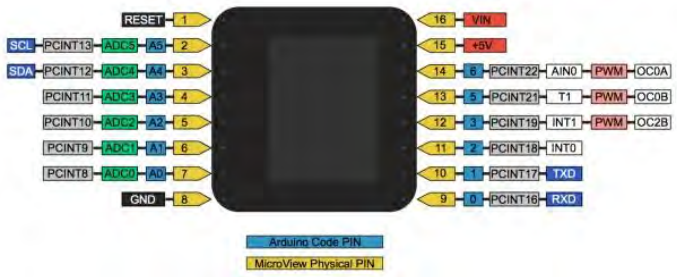
[16] PE, Isaac. Comparativa y análisis: Raspberry Pi vs competencia [en línea]. [Consultado 24/11/2015]. Disponible en internet: http://comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/

4.19 MODULO DE ARDUINO

4.19.1 MicroView. El MicroView es el primer módulo compatible con Arduino que permite ver lo que tu Arduino está pensando utilizando una pantalla incorporada OLED y que ha salido de una exitosa campaña de Kickstarter. Con una pantalla OLED de 64x48 píxeles a bordo, puedes utilizar el MicroView para mostrar datos de sensores, mensajes de correo electrónico, estado de pin, y más. La forma y pines del MicroView permite conectarlo directamente en una *protoboard* y hacer pruebas en pocos minutos. El MicroView también cuenta con una librería con todas las funciones de Arduino para hacer la programación del módulo muy fácil.

En el corazón de MicroView hay un microcontrolador ATmega328P, un regulador de 5V y 3.3V, una pantalla OLED monocromo de 64x48 píxeles, junto con otros componentes pasivos que permiten al MicroView funcionar sin otros componentes externos que no sean una fuente de alimentación. Además, el MicroView es 100% compatible con Arduino Uno (versión ATmega328P), es decir, el código que se ejecuta en un Arduino Uno también será capaz de correr en la MicroView si los pines IO utilizados en el código están expuestas externamente en el MicroView.[17]

Ilustración 18. Esquema funcional de la microview.



Fuente: SPARKFUN, Start something, MicroView Overview [en línea]. Colorado, 2012 [Consultado 10/03/2017]. Disponible en internet: https://learn.sparkfun.com/tutorials/sparkfun-inventors-kit-for-microview/microview-overview-

[17] SPARKFUN, Start something, MicroView Overview [en línea]. Colorado, 2012 [Consultado 10/03/2017]. Disponible en internet: https://learn.sparkfun.com/tutorials/sparkfun-inventors-kit-for-microview/microview-overview-

5 ANTECEDENTES

Se han desarrollado con hardware libre los siguientes proyectos implementando redes neuronales:

5.1 "LEUTERIO EL REMINISCENTE"

Diego Restrepo y Nelson Cárdenas, estudiantes del Programa de Ingeniería Electrónica de la Universidad del Magdalena (Santa Marta, Colombia) presentaron en *campus party* 2014 (Cali, Colombia) un proyecto denominado "Leuterio El Reminiscente" en el cual lograron implementar redes neuronales en hardware libre (Arduino). Para tal fin, desarrollaron el popular juego del "Triqui" (Tres en línea), a través de una interfaz en *processing* y entrenaron la red neuronal para que se enfrentara a la mente humana.[18]

Ilustración 19. Interfaz gráfica *processing*

Fuente: RESTREPO, Diego. CÁRDENAS, Nelson. Redes neuronales implementadas en Arduino [en línea]. [Consultado 21/11/2015]. Disponible en internet: http://www.prenser.com/3594/Redes_Neuronales_Implementadas_en_Ardui.html

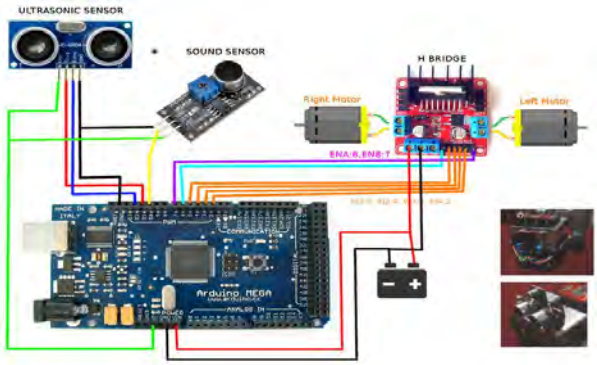
[18] RESTREPO, Diego. CÁRDENAS, Nelson. Redes neuronales implementadas en Arduino [en línea]. [Consultado 21/11/2015]. Disponible en internet: http://www.prenser.com/3594/Redes_Neuronales_Implementadas_en_Ardui.html

5.2 ROBOT EN ARDUINO QUE EMULA EL SISTEMA NERVIOSO DEL GUSANO NEMATODO UTILIZANDO REDES NEURONALES

Este proyecto es acerca de la creación de un robot en arduino, la cual por medio de redes neuronales emula el sistema nervioso del gusano nematodo C. Elegans que es el sistema que finalmente procesa la información y el cual envía las señales de movimiento al robot, es decir que el robot no se mueve por una programación del usuario sino que sus comportamientos emergen debido a la actividad neuronal simulada por el programa y que es estimulada por las neuronas receptoras o motoras que están conectados a los periféricos del robot.

El robot no tiene movimientos impresionantes, recordemos que está siendo operado por el sistema nervioso más simple que se conoce, solo 302 neuronas interconectadas gobiernan los comportamientos del *robot*, y esos comportamientos están condicionados por los estímulos de entrada, o sea el sensor ultrasónico que dispara la sinapsis de las neuronas ligadas a la nariz del gusano (lo que en un ambiente real seria como si el gusano chocara contra algo) y el sensor de sonido que dispara las neuronas relacionadas con el alimento (en un ambiente real seria como si el gusano encontrara rastros de alimentos).[19]

Ilustración 20. Conexión arduino/periféricos



Fuente: OLMOS, Fernando. Robot gusano con Arduino, paso a paso", [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://www.taringa.net/post/hazlo-tu-mismo/18508761/Robot-gusano-con-arduino-paso-a-paso.html

[19] OLMOS, Fernando. Robot gusano con Arduino, paso a paso", [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://www.taringa.net/post/hazlo-tu-mismo/18508761/Robot-gusano-con-arduino-paso-a-paso.html

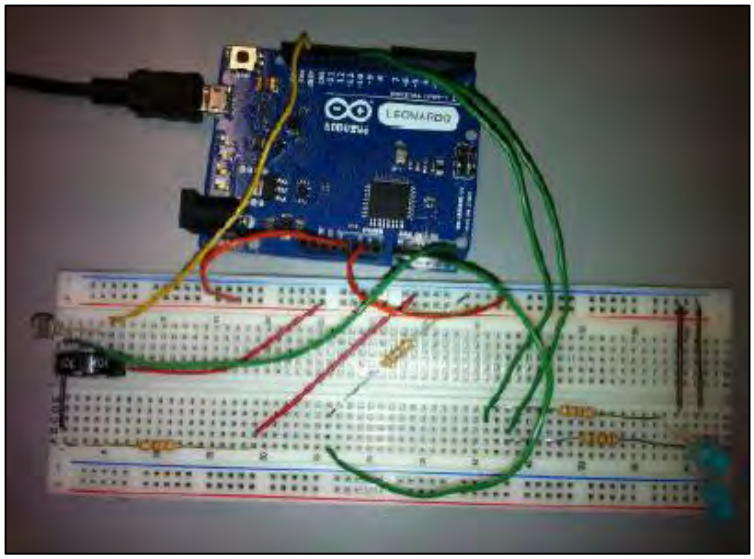
5.3 CONTROL DE ACCESO INTELIGENTE BASADO EN HARDWARE DE BAJO COSTE (ARDUINO LEONARDO) UTILIZANDO VISUAL BASIC

Para la elaboración de este proyecto, se ha realizado una aplicación en *visual basic* que procesa la imagen de entrada del vehículo mediante el software de tratamiento de imágenes Adobe Photoshop, de forma que extraiga la matrícula y se la pase a un motor OCR mediante acceso por línea de comandos. Uno de los motores OCR de software libre que podría realizar esta función es Tesseract, el cual tiene soporte para varios sistemas operativos, y una amplia trayectoria, por lo que es un sistema muy probado. Por su parte, Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo propio, programable en lenguajes como C, y teniendo módulos para interacción con MATLAB o Visual Basic.

Una de los principales retos de la inteligencia artificial, las redes neuronales y la lógica difusa es crear un sistema artificial que sea capaz de "ver" objetos e individuos y sacar conclusiones efectivas de ello. La visión por computador, por tanto, puede definirse como la capacidad de un sistema informático de extraer información de la realidad a partir de imágenes. En nuestro caso, este es el área de la inteligencia artificial que está más relacionada con nuestra operación, ya que el reconocimiento de matrículas se realizará por reconocimiento de patrones y definición gradual de coincidencias. Estrechamente ligada a esta disciplina está el procesamiento de imágenes, dedicado a mejorar la calidad de una fotografía obtenida a fin de que sea más fácilmente reconocible por el sistema, los gráficos por computador o el aprendizaje y razonamiento cognitivo, entre otros.[20]

[20] FERNÁNDEZ PAZ, Yago. Control de acceso inteligente basado en hardware de bajo costo [en línea]. [Consultado 23/11/2015]. Disponible en internet: ruc.udc.es/bitstream/2183/14433/2/FernandezPaz_Yago_TFG_2014.pdf.

Ilustración 21. Montaje placa Arduino.



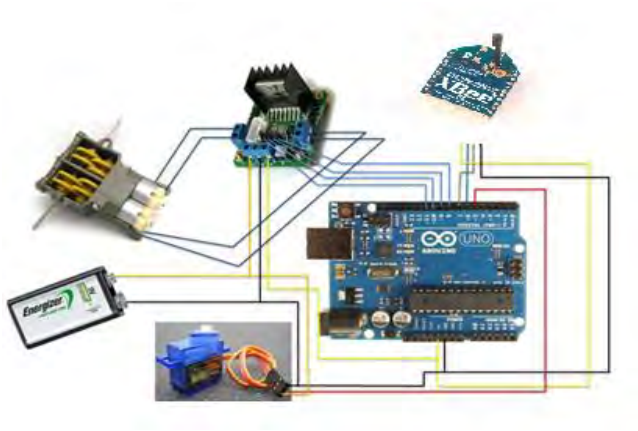
Fuente: FERNÁNDEZ PAZ, Yago. Control de acceso inteligente basado en hardware de bajo costo [en línea]. [Consultado 23/11/2015]. Disponible en internet: ruc.udc.es/bitstream/2183/14433/2/FernandezPaz_Yago_TFG_2014.pdf.

5.4 ROBOT EVASOR DE OBSTÁCULOS CON RED NEURONAL EN UN ARDUINO UNO

Este robot programado en un arduino uno, tiene una red neuronal embebida, previamente entrenada en Matlab con el fin de evitar los obstáculos que encuentre el robot, los sensores que lleva incorporados son unos sensores Sharp de 150 cm infrarrojos la señales son leídas a través de las entradas analógicas del arduino, las cuales vienen siendo las entradas de la red neuronal programada en el arduino. El *robot* cuenta con un módulo Xbee el cual envía los datos de los sensores y de sus movimientos a un PC, y son visualizados en un programa desarrollado en Visual Basic 6.0.[21]

[21] RÍOS, Edwin. Robot evasor de obstáculos de red neuronal en un Arduino [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://edwinri.blogspot.com.co/2013/06/robot-evasor-de-obstaculos-con-red.html

Ilustración 22. Conexión del Arduino periféricos.



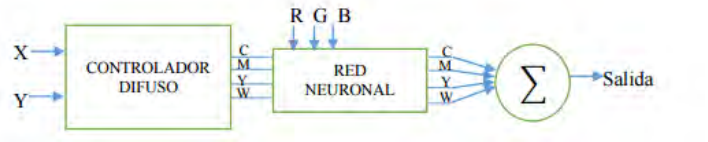
Fuente: VÉLEZ, Martin. Arma tu robot evasor de obstáculos (Arduino) [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://www.taringa.net/post/hazlo-tu-mismo/17753115/Arma-tu-Robot-evasor-de-obstaculos-Arduino.html.

5.5 DISEÑO DE UN PROTOTIPO NEURO-DIFUSO PARA LA PREPARACIÓN DE PINTURA A PARTIR DE LOS COLORES PRIMARIOS

Un sistema neuro-difuso es la combinación entre un sistema lógico difuso y redes neuronales dando lugar a un sistema inteligente con un razonamiento como lo hace el cerebro humano en la cual ha sido diseñado un sistema con redes neuronales para la clasificación de los colores primarios así también un controlador difuso para determinar los componentes de los colores en cuestión con la finalidad de replicar un color de la plantilla basada en el arcoíris de Granger. A partir de las coordenadas de las mismas. Los requerimientos en cuanto a software y hardware que cumple los requisitos para la realización de este proyecto es la Raspberry Pi, como este, existe otros sistemas embebidos para el desarrollo de aplicaciones, así también hay diferentes versiones del dispositivo en cuestión, particularmente corresponde al modelo B, el cual permite alojar periféricos de entrada y salida.[22]

[22] PULLA SÁNCHEZ, Víctor. Diseño de un prototipo Neuro-Difuso para la preparación de pintura a partir de los colores primarios, Cian, Magenta y amarillo (CMY) para el coloreado de artesanías en paja y cerámica [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://dspace.ups.edu.ec/bitstream/123456789/8288/1/UPS-CT004926.pdf

Ilustración 23. Sistema Neuro-Difuso



Fuente: PULLA SÁNCHEZ, Víctor. Diseño de un prototipo Neuro-Difuso para la preparación de pintura a partir de los colores primarios, Cian, Magenta y amarillo (CMY) para el coloreado de artesanías en paja y cerámica [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://dspace.ups.edu.ec/bitstream/123456789/8288/1/UPS-CT004926.pdf

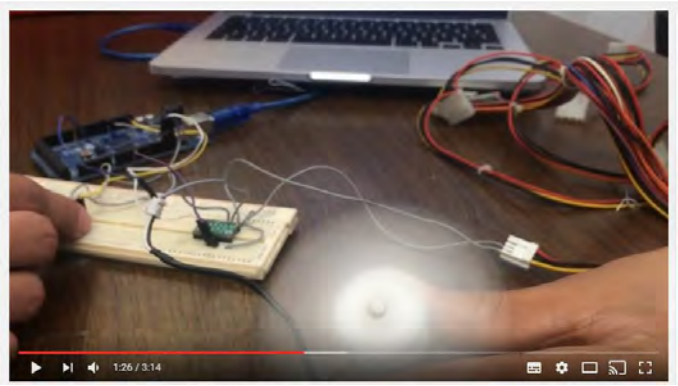
5.6 DISEÑO DE CONTROL INTELIGENTE (DIFUSO) LM35+VENTILADOR

Este diseño fue realizado, por el estudiante Luis Alor, estudiante de la universidad politécnica de Chiapas (México), para la presentación del proyecto de corte No.3 de la materia control inteligente. Consiste en control difuso, implementado con el sensor LM35 y un motor, el cual simula un ventilador, también la conexión arduino -matlab. El cual según las lecturas obtenidas por el sensor LM35, el ciclo de motor aumenta o disminuye, de acuerdo a la escala de temperatura que se encuentre, por ejemplo, si está a temperatura ambiente (25°C), el ciclo de trabajo del motor es del 9%-18%, lo que significa que incrementa la velocidad del motor y si aumenta la temperatura hasta 40°C, el ciclo de trabajo del motor es alrededor del 30%- 40%, nos indica que aumenta casi llegando a su límite de velocidad.

Todo esto funciona con lógica difusa, del cual se crea un control inteligente utilizando las herramienta que brinda el *toolbox* Fuzzy *Logic*, de matlab, en los cuales se tienen los conjuntos difusos y sus respectivas funciones de pertenencia (frio, fresco, caluroso, muy caluroso y calor extremo), cada uno tiene un rango de valores de los que incluye el conjunto y la salida es el ciclo de trabajo del pwm del motor el cual incluye las funciones de pertenencia(apagado, muy lento, medio lento, rápido y muy rápido) igual con los valores de los conjuntos con diferente valores del pwm del motor. Para el diseño del panel de control se utilizó la herramienta de Matlab (GUI-*building tool*), se creó el código, mediante la programación enfocada a objetos, y la configuración de conexiones con arduino, todo esto fue realizado en el sistema operativo Mac OS X.

En la ilustración 24 se muestra un pantallazo del video, y su referencia del video que se encuentra en YouTube.

Ilustración 24. Control inteligente LM35 + ventilador



Fuente: ALOR, Luis. Control inteligente LM35+ventilador [en línea]. Chiapas: Universidad de Chiapas, 2014 [Consultado 11/03/2017]. Disponible en internet: https://www.youtube.com/watch?v=7j5MYo9I0l8

6 PRESENTACIÓN DE LAS HERRAMIENTAS IMPLEMENTADAS

6.1 HERRAMIENTA No.1: IMPLEMENTACIÓN DE UNA RED NEURONAL PARA EL RECONOCIMIENTO DE LOS NÚMEROS DEL 0-9

En esta primera implementación, consiste en mostrar por medio de una red neuronal de tipo perceptrón, el reconocimiento de los dígitos del 0 al 9. Mediante *dip-switch* se emula las posibles entradas que puede tener la red de tipo perceptrón simple para el respectivo reconocimiento de los números. Se considera utilizar una matriz con una dimensión de 5x3, para ello se debe caracterizar las diferentes combinaciones para cada uno de los números de 0 y el 9, donde 0 indica que no está activado y 1 cuando está activado este sensor, como se muestra respectivamente en la ilustración 25, con esto se va tener una matriz entrada de 15x10, el numero 15 corresponde a el número de *dip-switch* y el 10 los diferentes números de 0 al 9. Como se muestra en la ilustración 26.

Por medio de un *display* se va mostrar la salida de la red neuronal, pero se debe realizar una matriz de 4x10, donde las 4 filas corresponde a la codificación BCD para la visualización del *display* y las 10 columnas corresponde a los números del 0 al 9, como se muestra en la ilustración No. 27 que va ser el número que identifica los sensores mediante el entrenamiento que tiene la red neuronal, pero primero se debe entrenar o implementar en Matlab para mayor facilidad en el procesamiento de los diferentes de los pesos, para luego respectivamente normalizarlos e implementarlo en la plataforma de arduino Mega, que es la plataforma donde finalmente se realizara, ya que en el mercado actual es la plataforma de hardware libre, para el objetivo educativo para el aprendizaje de redes neuronales de una forma práctica y de fácil uso.

Ilustración 25. Codificación de los *dip-Stwich* con los diferentes números de 0 al 9.

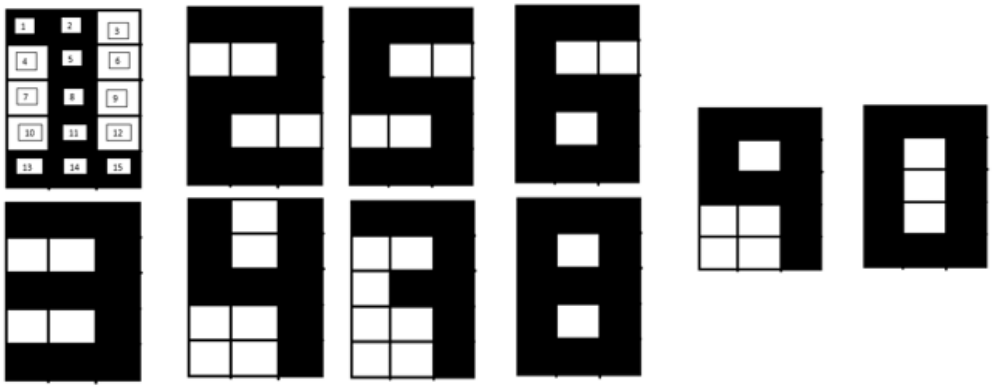


Ilustración 26. Matriz 15x10 de las salidas de los *dip-switch*(derecha). Matriz 4x10 de codificación BCD del *display.*

$$x = \begin{bmatrix} 1&1&1&1&1&1&1&1&1&1 \\ 1&1&1&0&1&1&1&1&1&1 \\ 0&1&1&1&1&1&1&1&1&1 \\ 0&0&0&1&1&1&0&1&1&1 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&1&1&1&0&0&1&1&1&1 \\ 0&1&1&1&1&1&0&1&1&1 \\ 1&1&1&1&1&1&1&1&1&0 \\ 0&1&1&1&1&1&1&1&1&1 \\ 0&1&0&0&0&1&0&1&0&1 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&1&1&1&1 \\ 1&1&1&0&1&1&0&1&0&1 \\ 1&1&1&0&1&1&0&1&0&1 \\ 1&1&1&1&1&1&1&1&1&1 \\ \downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow \\ 1&2&3&4&5&6&7&8&9&0 \end{bmatrix}$$

$$T = \begin{matrix} S3 \\ S2 \\ S1 \\ S0 \end{matrix} \begin{bmatrix} 0&0&0&0&0&0&0&1&1&0 \\ 0&0&0&1&1&1&1&0&0&0 \\ 0&1&1&0&0&1&1&0&0&0 \\ 1&0&1&0&1&0&1&0&1&0 \\ \downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow&\downarrow \\ 1&2&3&4&5&6&7&8&9&0 \end{bmatrix}$$

Ilustración 27. Diagrama de flujo de la implementación.

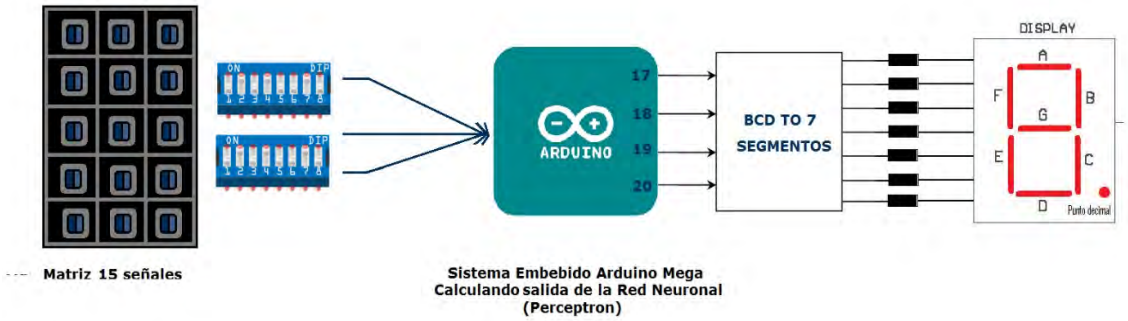
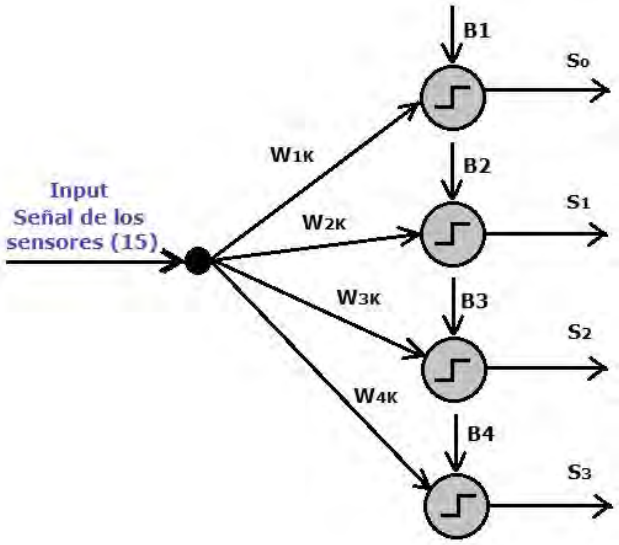


Ilustración 28. Diagrama de la red neuronal tipo perceptrón que se va implementar.



En la ilustración 28 se puede apreciar el tipo de red neuronal que se va implementar mediante el software de Matlab, esta red es de tipo perceptrón de una sola capa, con cuatro neuronas respectivamente, ya que son 4 salidas. La función de activación es de tipo escalón (hardlim), la salida de la neurona es cero, si la neta es menor que cero y si la salida de la neurona es 1, siempre que la neta sea mayor a cero. En la ecuación se muestra la salida, y el cálculo de la Neta. Para poder generar la red en el sistema embebido arduino Mega es necesario primero realizamos el entrenamiento de la red, por la cual se permite calcular la matriz de pesos, esto es necesario para el cálculo de la neta de cada neurona. Además, permite obtener la matriz correspondiente al byas de cada neurona.

$$Salida = \begin{Bmatrix} 0 \rightarrow \ N_{eta} \geq 0 \\ 1 \rightarrow \ N_{eta} < 0 \end{Bmatrix}$$

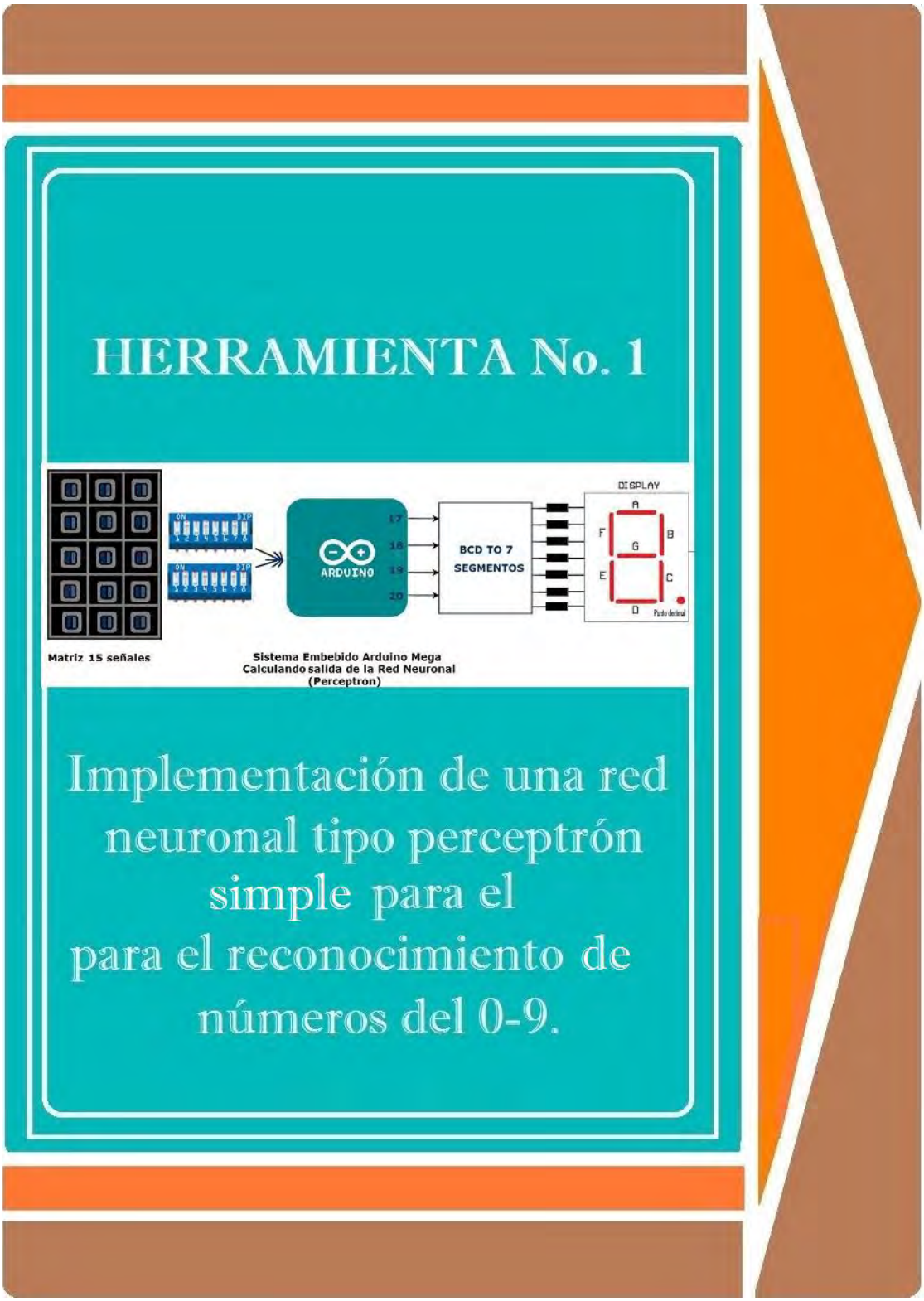
$$Neta_k = \left(\sum_{i=0}^{j} W_{jk} P_j \right) + b_x$$

$\rightarrow \ W_{jk}: matriz\ de\ Pesos;\ P_j: Entrada\ de\ la\ red; b_x: Byas(umbral\ de\ activación)$

Ecuación No.1 Ecuación de la salida de la red neuronal y el cálculo respectivo de la Neta

Ya conociendo previamente la descripción, se mostrará a continuación la guía de la herramienta No.1 del procedimiento realizado paso a paso:

Ilustración 29. Portada de herramienta 1



• **Objetivo.** Conocer cómo se puede aplicar una red neuronal tipo *Perceptron* en un problema de reconocimiento de patrones.

Implementar una red neuronal tipo perceptrón simple para realizar el proceso de aprendizaje y reconocimiento de caracteres numéricos (0-9).

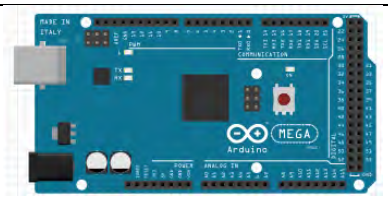
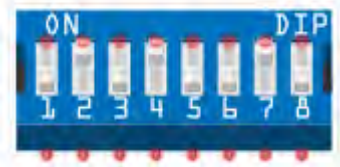
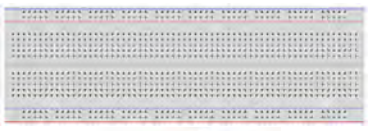
Utilizar Arduino como plataforma de implementación de redes neuronales.

• **Descripción del proyecto.** Esta primera implementación, consiste en mostrar por medio de una red neuronal de tipo perceptrón, el reconocimiento de los dígitos del 0 al 9, mediante de *dip-switch*, se emula las posibles entradas que puede tener la red de tipo perceptrón simple para el respectivo reconocimiento de los números, se considera utilizar una matriz con una dimensión de 5x3, para ello se debe caracterizar las diferentes combinaciones para cada uno de los números de 0 y el 9, donde 0 indica que no está activado y 1 cuando está activado, con esto se va tener una matriz entrada de 15x10, el numero 15 corresponde a el número de *dip-switch* y el 10 los diferentes números de 0 al 9.

Por medio de un *display* se va mostrar la salida de la red neuronal, pero se debe realizar una matriz de 4x10, donde las 4 filas corresponde a la codificación BCD para la visualización del *display* y las 10 columnas corresponde a los números del 0 al 9, que va ser el número que emulan los *dip-switch* mediante el entrenamiento que tiene la red neuronal, pero primero se debe entrenar o implementar en Matlab para mayor facilidad en el procesamiento de los diferentes de los pesos, para luego respectivamente normalizarlos e implementarlo en la plataforma de Arduino Mega.

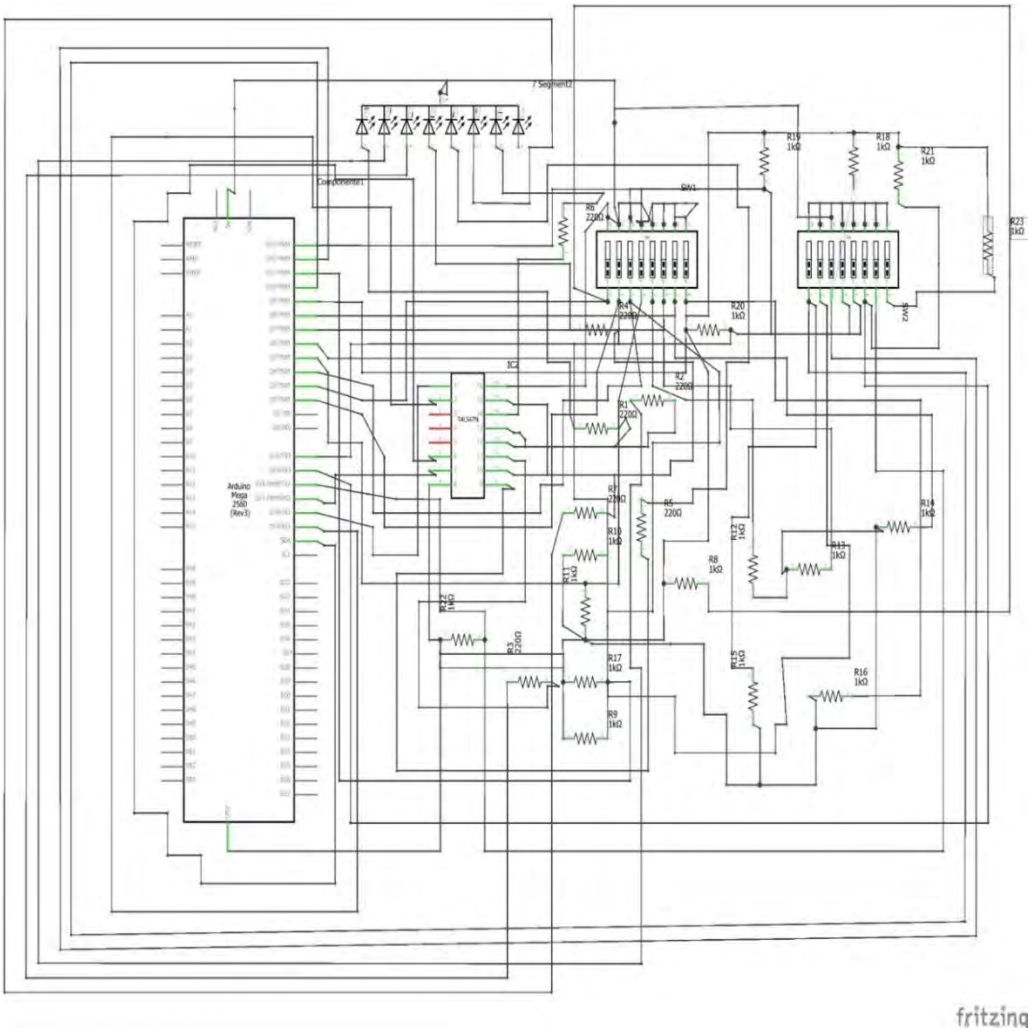
- **Materiales requeridos**

Tabla 1. Materiales Herramienta No.1

	ARDUINO MEGA
	DISPLAY 7 SEGMENTOS
	7 RESISTENCIAS DE 220 Ω
	16 RESISTENCIS DE 1Ω
	DECODIFICADOR BCD A 7 SEGMENTOS
	2 DIP SWITCH 8 SEGMENTOS
 	UNA PROTOBOARD Y CABLES DE CONEXIÓN

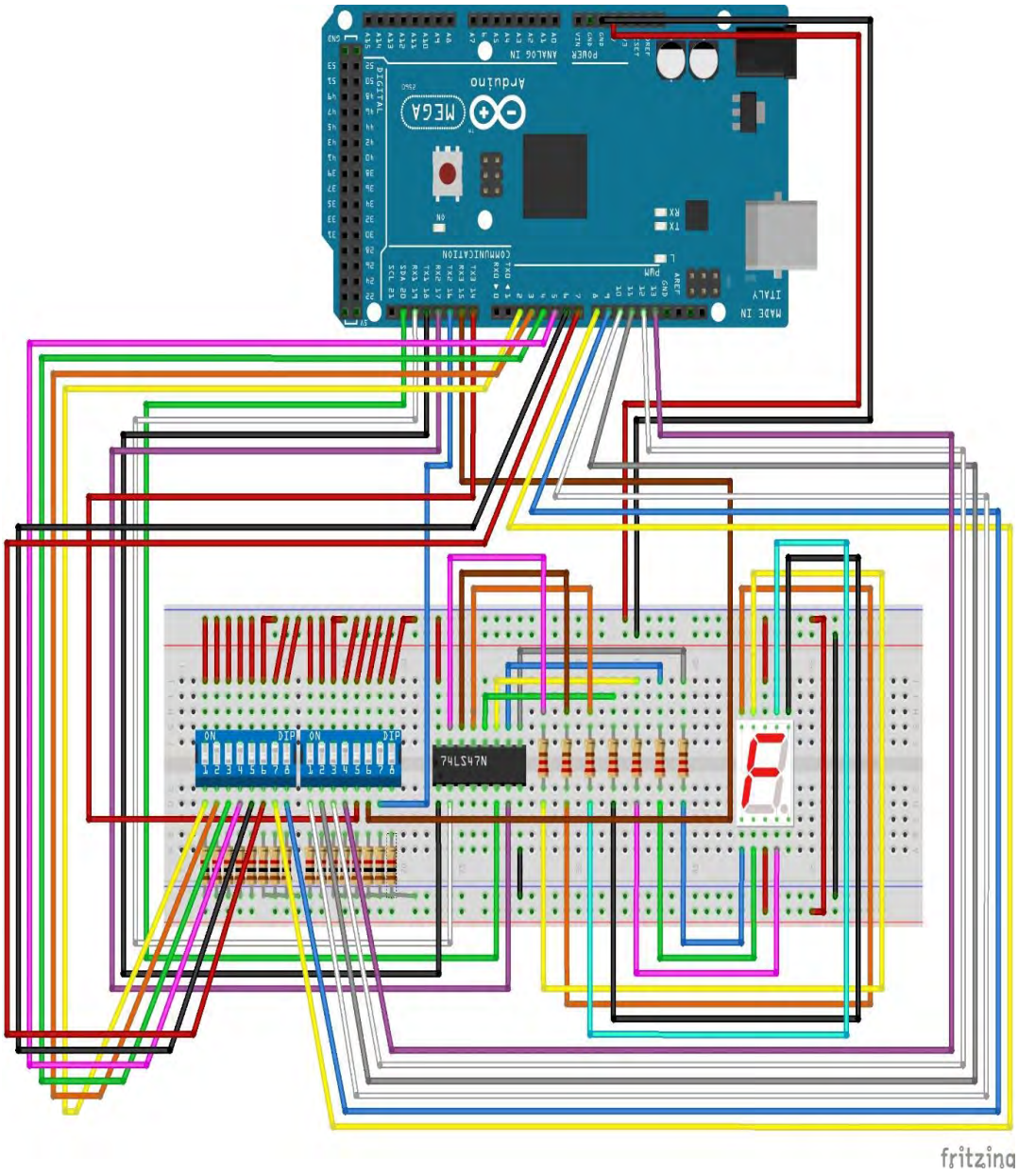
• **Esquema electrónico del circuito.** En la ilustración 29 se muestra el diagrama esquemático del circuito, que genera la herramienta Fritzing.

Ilustración 30. Diagrama esquemático.



Y el circuito en la protoboard, se puede visualizar por medio de la herramienta Fritzing, que nos facilita la visualización de cada una de las conexiones, como se muestra en la ilustración 30.

Ilustración 31. Conexiones con Arduino Mega y materiales requeridos.



- **Código. Entradas de la red.**

En esta parte del código se van a mostrar las entradas de la red, que son la del sensor CNY70, que se emulan con Dip-switch.

```
int cny1 = 2;

int cny2 = 3;

int cny3 = 4;

int cny4 = 5;

int cny5 = 6;

int cny6 = 7;

int cny7 = 8;

int cny8 = 9;

int cny9 = 10;

int cny10 = 11;

int cny11 = 12;

int cny12 = 13;

int cny13 = 14;

int cny14 = 15;
```

```
int cny15 = 16;
```

Estas son las salidas de la red

```
int sal1 = 20;

int sal2 = 19;

int sal3 = 18;

int sal4 = 17;
```

Configuración del modo de entrada o salida.

```
void setup() {
  pinMode(cny1,INPUT);
  pinMode(cny2,INPUT);
  pinMode(cny3,INPUT);
  pinMode(cny4,INPUT);
  pinMode(cny5,INPUT);
  pinMode(cny6,INPUT);
  pinMode(cny7,INPUT);
  pinMode(cny8,INPUT);
  pinMode(cny9,INPUT);
```

```
  pinMode(cny10,INPUT);
  pinMode(cny11,INPUT);
  pinMode(cny12,INPUT);
  pinMode(cny13,INPUT);
  pinMode(cny14,INPUT);
  pinMode(cny15,INPUT);
  pinMode(sal1,OUTPUT);
  pinMode(sal3,OUTPUT);
  pinMode(sal2,OUTPUT);
  pinMode(sal4,OUTPUT);
  Serial.begin(9600);
}
void loop() {
```

Si la neta es mayor a 0, la salida = 1

De lo contrario la salida = 0

******En esta parte de realiza la lectura de las entradas de la Red Neuronal*******

```
int Vi1=digitalRead(cny1);int Vi2=digitalRead(cny2);int Vi3=digitalRead(cny3);int Vi4=digitalRead(cny4);int Vi5=digitalRead(cny5);

 int Vi6=digitalRead(cny6);int Vi7=digitalRead(cny7);int Vi8=digitalRead(cny8);int Vi9=digitalRead(cny9);int Vi10=digitalRead(cny10);

 int Vi11=digitalRead(cny11);int Vi12=digitalRead(cny12);int Vi13=digitalRead(cny13);int Vi14=digitalRead(cny14);int Vi15=digitalRead(cny15);
```

Calculo de la función de salida del perceptrón simple y con los pesos ya calculados

```
 double Neta1=(-6*Vi1+2*Vi2-3*Vi3+6*Vi4-3*Vi5+1*Vi6-3*Vi7+12*Vi8-3*Vi9+2*Vi10-3*Vi11+6*Vi12-1*Vi13-1*Vi14-6*Vi15)-6;

 double Neta2=(3*Vi1-10*Vi2+5*Vi3-2*Vi4-2*Vi5-13*Vi6-3*Vi7+3*Vi8+5*Vi9-5*Vi10-2*Vi11+5*Vi12-1*Vi13-1*Vi14-3*Vi15)+3;

 double Neta3=(0*Vi1+2*Vi2+4*Vi3-19*Vi4-4*Vi5-7*Vi6+4*Vi7+0*Vi8+4*Vi9+6*Vi10-4*Vi11+0*Vi12+2*Vi13+2*Vi14+0*Vi15)+0;

 double Neta4=(0*Vi1+1*Vi2+0*Vi3-1*Vi4+0*Vi5+0*Vi6-1*Vi7+0*Vi8+0*Vi9-2*Vi10+0*Vi11+1*Vi12+0*Vi13+0*Vi14+0*Vi15)+0;
```

Se realiza la generación de la salida dela red, las diferentes condiciones, para el reconocimiento del carácter numérico

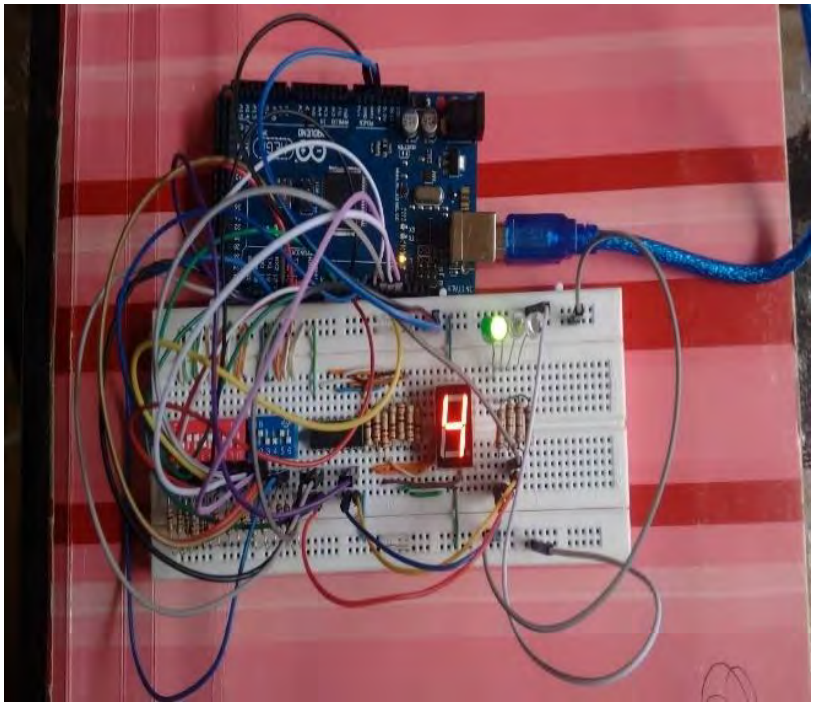
```
 if(Neta1>=0){

   digitalWrite(sal1,HIGH);

 }
```

```
if(Neta1<0){
  digitalWrite(sal1,LOW);
}
if(Neta2>=0){
  digitalWrite(sal2,HIGH);
}
if(Neta2<0){
  digitalWrite(sal2,LOW);
}
if(Neta3>=0){
  digitalWrite(sal3,HIGH);
}
if(Neta3<0){
  digitalWrite(sal3,LOW);
if(Neta4>=0){
  digitalWrite(sal4,HIGH);
}
```

```
  if(Neta4<0){

    digitalWrite(sal4,LOW);

  }

 Serial.println("Neta1 Neta2 Neta3 Neta4");

  Serial.println(Neta1);

  Serial.println(Neta2);

  Serial.println(Neta3);

  Serial.println(Neta4);*/

  delay(1000);

}
```

- **Resultados.** Aquí tenemos unas imágenes del circuito montado en la protoboard, conectado al Arduino y funcionando. los diswicth hace parte de la enmulacion de la matrix 5x 3 de senores CNY 70, por lo tanto se debe tener en cuenta la codificacion como se muestra en la ilustración 31. Ejemplo de como seria la codificacion. los pines del 1 al 15 van a emular las salida del sensor que se convierten en las entradas del arduino. tambien podemos observar cuatro leds que van a simular los numeros binarios siendo el primer led de la derecha el menos significativo y el cuarto el mas significativo, esto quiere decir que como vemos la imagen se esta simulando el numero cuatro, el tercer led esta prendido ya que en binario el número simulado es 0100. En la segunda imagen se simula el numero cero, por lo tanto ningun led esta prendido.

Ilustración 32. Funcionamiento real de la herramienta.

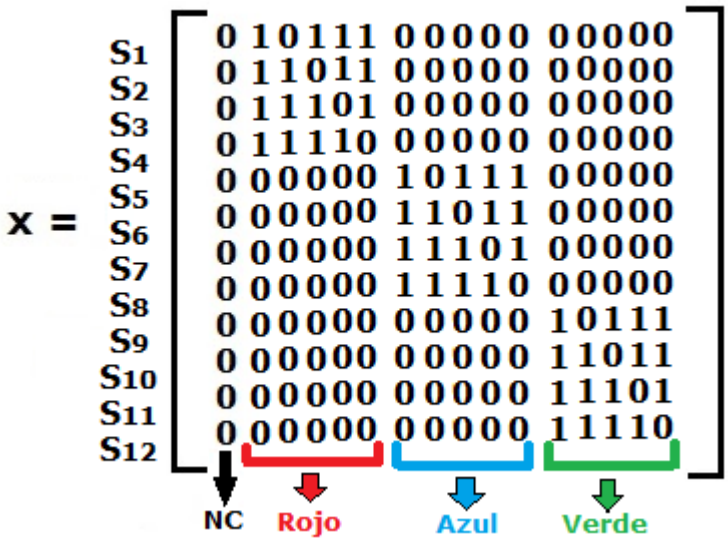


6.2 HERRAMIENTA No.2: IMPLEMENTACIÓN DE UN PERCEPTRON PARA DISTINCION DEL COLOR PREDOMINANTE (ROJO-VERDE-AZUL)

En este capítulo se describe como fue la implementación de un *Perceptron* para la distinción de colores emulando por medio de *dip-switch*, se emula las posibles entradas que puede tener la red de tipo perceptrón simple para el respectivo reconocimiento del color predominante, se implementa por medio de una red neuronal de tipo perceptrón multicapa, el cual permite obtener tres salidas de acuerdo al color que el sensor detecte (se activa la salida de acuerdo al color que predomine).

Continuación se muestra la matriz de las diferentes salidas que puede tener el sensor (Rojo, azul y verde):

Ilustración 33. Matriz de salida



S1-S4 corresponden a la salida del sensor rojo, S5-S8 a las salidas del sensor azul y S9-S12 a la salida del sensor verde. La matriz de entrada de la red consta de 12 filas que corresponden a las señales generadas por los sensores de color, y 16 columnas el cual corresponden a las diferentes combinaciones de los sensores que se pueden obtener para la caracterización.

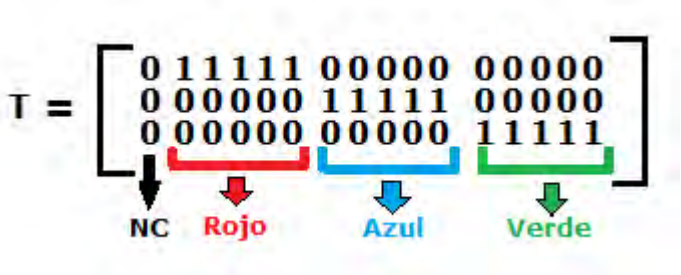
Por ejemplo, si el objeto es de color azul las combinaciones pueden ser:

- Que todos los sensores arrojen salida activa en azul.

• Por lo menos un sensor no arroje la salida activa en azul.

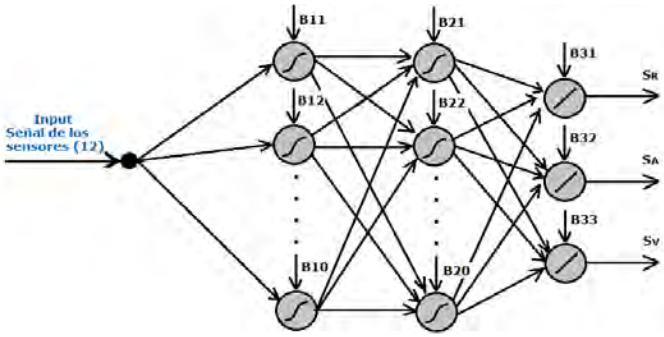
De igual manera se reflejará estas condiciones con los otros dos colores rojo y verde. Para la salida de la red se requiere 3 neuronas, debido a que se quiere caracterizar solo 3 colores. La matriz posee 3 filas (posibles colores detectado) y 16 columnas (respuesta del color a las diferentes combinaciones que se presenta en la entrada). Se muestra a continuación la matriz de la salida de la red:

Ilustración 34. Matriz de salida de la red



En este caso el perceptrón multicapa se caracteriza porque tiene sus neuronas agrupadas en capas de diferentes niveles. Cada una de las capas están formada por un conjunto de neuronas y se distinguen tres tipos de capas diferentes: la capa de entrada, las capas ocultas y la capa de salida, en nuestro caso la red neuronal corresponde a un perceptrón multicapa, cada capa contiene 10 neuronas de entrada la cual no actúa como neuronas propiamente dichas, sino que se encargan únicamente de recibir las señales o patrones del exterior y propagar dicha señal a las neuronas de la siguiente capa que son 3 neuronas en la capa de salida.

Ilustración 35. Perceptrón multicapa de la herramienta 2.



Ya conociendo previamente la descripción, se mostrará a continuación la guía de la herramienta No.2 del procedimiento realizado paso a paso:

Ilustración 36. Portada herramienta 2

• **Objetivos.** Conocer cómo se puede aplicar una red neuronal tipo *Perceptron* multicapa en un problema de reconocimiento de colores RGB.

Implementar una red neuronal tipo *Perceptron* multicapa para realizar el proceso de aprendizaje y reconocimiento de colores.

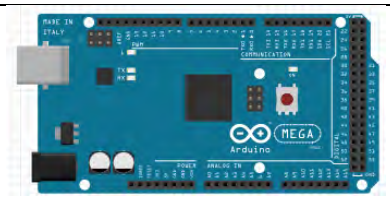
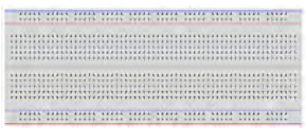
Utilizar Arduino como plataforma de implementación de redes neuronales *Perceptron* multicapa

• **Descripción del proyecto.** En este proyecto se describe como fue la implementación de un perceptrón multicapa para la distinción de colores, se emula por medio de *dip-switch* las entradas de la red multicapa, el cual permite obtener tres salidas de acuerdo al color censado (se activa la salida de acuerdo al color).

En este caso la red neuronal corresponde a un perceptrón multicapa, cada capa contiene 10 neuronas de entrada la cual no actúa como neuronas propiamente dichas, sino que se encargan únicamente de recibir las señales o patrones del exterior y propagar dicha señal a las neuronas de la siguiente capa que son 3 neuronas en la capa de salida.

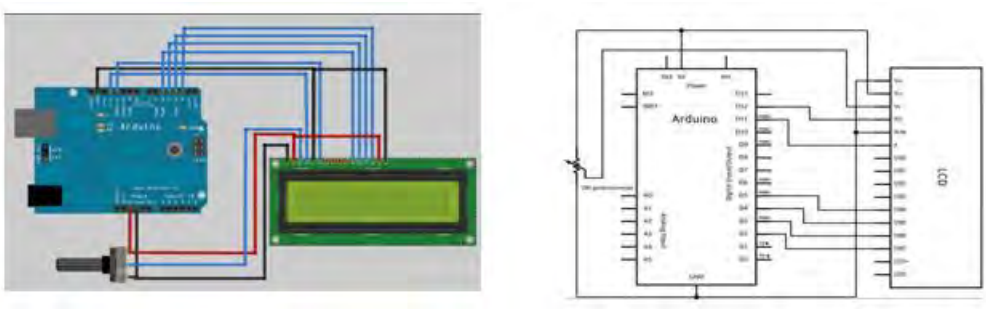
- **Materiales requeridos**

Tabla 2. Materiales herramienta No.2

	ARDUINO MEGA
	LCD 16X2
	15 RESISTENCIAS DE 220 Ω
	DIODO LED ROJO, AZUL Y VERDE
	UN POTENCIOMETRO DE 10KΩ
	6 DIP SWITCH 2 SEGMENTOS
 	UNA PROTOBOARD Y CABLES DE CONEXIÓN

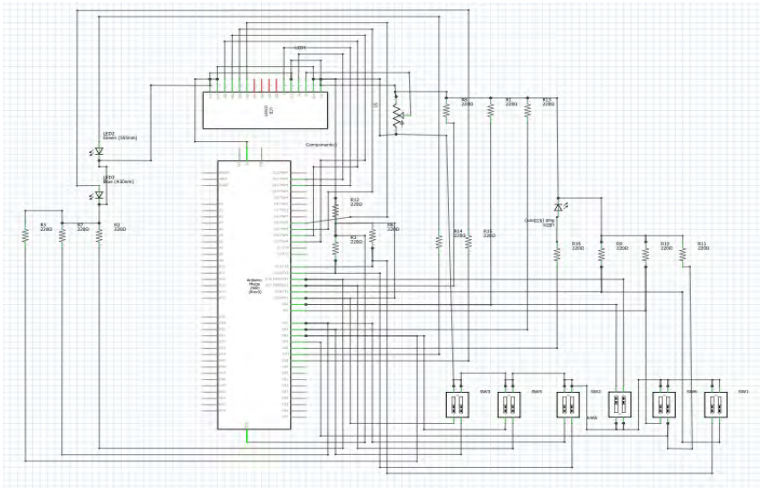
- **Esquema electrónico del circuito.** Vamos a montar un circuito que lea un color ya sea rojo, azul o verde por medio de dip- switch entrenando previamente la red neuronal y se leerá por medio de una LCD 16x2. Para conectar la LCD a un arduino independientemente, incluyo los detalles de sus pines en la ilustración 35.

Ilustración 37. Conexiones de LCD en Arduino Mega.



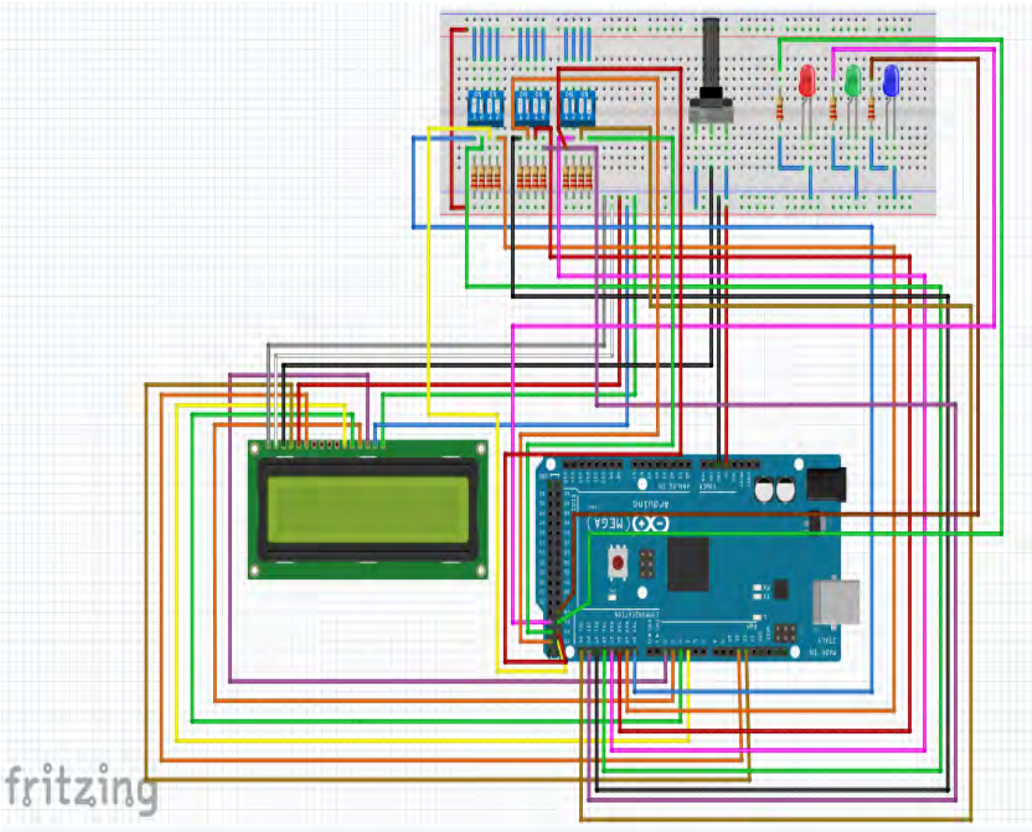
Aquí tenemos el esquema electrónico del circuito:

Ilustración 38. Diagrama esquemático.



Y el circuito en la protoboard, se puede visualizar por medio de la herramienta Fritzing, como se muestra en la ilustración 37.

Ilustración 39. Conexiones con Arduino Mega y materiales requeridos.



- **Código. Se implementa el código de caracterización de color predominante azul, rojo o verde implementando una red neuronal perceptrón multicapa.**

Se incluyen las librerías

```
#include <LiquidCrystal.h>

#include <math.h>

LiquidCrystal lcd(12,11,5,4,3,2);
```

Entradas digitales de las señales obtenidas.

```
int IR1=14;
```

```
int IR2=15;

int IR3=16;

int IR4=17;

int IG1=18;

int IG2=19;

int IG3=20;

int IG4=21;

int IB1=22;

int IB2=23;

int IB3=24;

int IB4=25;

int SR=26;

int SG=27;

int SB=28;

int cont=0;

double NetaCapa1[10];

double SalidaRed1[10];
```

```
double NetaCapa2[10];

double SalidaRed2[10];

double NetaCapaS[3];

double Salida[3];

void setup() {

  Serial.begin(9600);

  lcd.begin(16, 2);

pinMode(IR1,INPUT);pinMode(IR2,INPUT);pinMode(IR3,INPUT);pinMode(IR4,INP
UT);

pinMode(IB1,INPUT);pinMode(IB2,INPUT);pinMode(IB3,INPUT);pinMode(IB4,INP
UT);

pinMode(IG1,INPUT);pinMode(IG2,INPUT);pinMode(IG3,INPUT);pinMode(IG4,INP
UT);

  pinMode(SR,OUTPUT);pinMode(SG,OUTPUT);pinMode(SB,OUTPUT);

  lcd.print("MLP Arduino");

  lcd.setCursor(0,1);

  lcd.print("Config Pines");

  delay(2000);
```

```
  lcd.clear();

  lcd.setCursor(0,0);

  lcd.print("Reading Inputs");

  delay(2000);

  lcd.clear();

}

void loop() {

  int R1=digitalRead(IR1);int R2=digitalRead(IR2);int R3=digitalRead(IR3);int
R4=digitalRead(IR4);

  int G1=digitalRead(IG1);int G2=digitalRead(IG2);int G3=digitalRead(IG3);int
G4=digitalRead(IG4);

  int b1=digitalRead(IB1);int B2=digitalRead(IB2);int B3=digitalRead(IB3);int
B4=digitalRead(IB4);
```

Calculo de la neta con los pesos incluidos de la primera capa

```
  NetaCapa1[0]=(0.5873*R1-0.7269*R2+0.3642*R3+0.5384*R4-0.0296*G1-
0.3703*G2+0.3688*G3+0.5768*G4-0.4373*b1-0.6881*B2-0.7555*B3-0.4183*B4)-
0.6467;

  NetaCapa1[1]=(0.8032*R1+0.9328*R2-0.8700*R3-0.9130*R4-
0.1111*G1+0.2146*G2-0.5761*G3-0.1472*G4+0.7495*b1-0.7818*B2+0.7158*B3-
0.7368*B4)+0.8845;
```

```
  NetaCapa1[2]=(-0.650*R1+1.0103*R2+0.5973*R3-
0.2351*R4+0.3648*G1+0.5451*G2+0.2521*G3+0.2399*G4+0.2165*b1-0.0393*B2-
0.3629*B3-0.5826*B4)-0.8661;


  NetaCapa1[3]=(0.8511*R1-0.0049*R2+0.8608*R3-0.6351*R4+0.6288*G1-
0.5601*G2+0.4211*G3-
0.3351*G4+0.2879*b1+0.5971*B2+0.2480*B3+0.8420*B4)+0.6893;


  NetaCapa1[4]=(0.2291*R1+0.5649*R2+0.3369*R3-0.9582*R4-0.6960*G1-
0.7184*G2+0.7710*G3+0.6471*G4+0.7044*b1+0.7840*B2-0.8309*B3-
0.7731*B4)+0.6020;


  NetaCapa1[5]=(-0.866*R1-0.7775*R2+0.5339*R3+0.3844*R4-0.0123*G1-
0.0853*G2+0.8293*G3-0.4240*G4-0.6363*b1-0.7438*B2-
0.0075*B3+0.5528*B4)+0.6248;


  NetaCapa1[6]=(-0.526*R1-
0.2393*R2+0.6050*R3+0.2552*R4+0.0830*G1+0.6665*G2-0.1612*G3-
0.2078*G4+0.4748*b1+0.2876*B2-0.4528*B3+0.0913*B4)-0.9135;


  NetaCapa1[7]=(0.0345*R1+0.7722*R2-0.2136*R3-0.4864*R4+0.2415*G1-
0.2111*G2-0.7938*G3-0.8833*G4+0.2368*b1-0.0691*B2+0.0371*B3+0.6913*B4)-
0.1631;


NetaCapa1[8]=(0.9086*R1+0.5780*R2+0.2856*R3+0.6456*R4+0.3494*G1+0.0729
*G2-0.6574*G3+0.2030*G4-0.2771*b1-1.0960*B2+0.2956*B3-0.6856*B4)-0.6461;


NetaCapa1[9]=(0.8899*R1+0.8791*R2-0.6434*R3-1.0419*R4+0.5402*G1-
0.5826*G2-0.5537*G3-0.1607*G4-0.0139*b1-0.2869*B2+0.3570*B3-
0.2491*B4)+0.5667;
```

Salida Primera Capa Función de activación TanSig

```
  for(int i=0;i<=9;i=i+1){

     SalidaRed1[i]=NetaCapa1[i];
```

```
}
```

Calculo de la neta con los pesos incluidos de la segunda capa

```
  NetaCapa2[0]=(-
0.0291*SalidaRed1[0]+0.4341*SalidaRed1[1]+0.0244*SalidaRed1[2]+0.6718*Salid
aRed1[3]-0.3798*SalidaRed1[4]-
0.1190*SalidaRed1[5]+0.6582*SalidaRed1[6]+0.3424*SalidaRed1[7]-
0.5795*SalidaRed1[8]-0.2928*SalidaRed1[9])+0.1449;


  NetaCapa2[1]=(0.7343*SalidaRed1[0]+0.1148*SalidaRed1[1]-
0.8840*SalidaRed1[2]-0.3067*SalidaRed1[3]-0.2205*SalidaRed1[4]-
0.4624*SalidaRed1[5]+0.4658*SalidaRed1[6]-0.3439*SalidaRed1[7]-
0.2089*SalidaRed1[8]+0.8054*SalidaRed1[9])-0.3760;


  NetaCapa2[2]=(-0.7023*SalidaRed1[0]-
0.2389*SalidaRed1[1]+0.8200*SalidaRed1[2]-
0.5772*SalidaRed1[3]+0.6039*SalidaRed1[4]+0.5034*SalidaRed1[5]+0.0876*Salid
aRed1[6]+0.4946*SalidaRed1[7]-0.0653*SalidaRed1[8]-
0.1624*SalidaRed1[9])+0.2041;


  NetaCapa2[3]=(-
0.6124*SalidaRed1[0]+0.1073*SalidaRed1[1]+0.7601*SalidaRed1[2]-
0.3191*SalidaRed1[3]-1.0043*SalidaRed1[4]-0.4981*SalidaRed1[5]-
0.0632*SalidaRed1[6]-0.0214*SalidaRed1[7]-0.4004*SalidaRed1[8]-
0.7029*SalidaRed1[9])+0.5262;


  NetaCapa2[4]=(-0.6555*SalidaRed1[0]-
0.2403*SalidaRed1[1]+0.0455*SalidaRed1[2]-0.7407*SalidaRed1[3]-
0.8884*SalidaRed1[4]+0.3232*SalidaRed1[5]-0.1238*SalidaRed1[6]-
0.2628*SalidaRed1[7]+0.6199*SalidaRed1[8]+0.8509*SalidaRed1[9])-0.6104;


  NetaCapa2[5]=(-0.7765*SalidaRed1[0]-0.7444*SalidaRed1[1]-
0.1008*SalidaRed1[2]-0.7946*SalidaRed1[3]-0.7221*SalidaRed1[4]-
0.5533*SalidaRed1[5]-0.3191*SalidaRed1[6]+0.8456*SalidaRed1[7]-
0.5912*SalidaRed1[8]+0.9180*SalidaRed1[9])-0.7349;


  NetaCapa2[6]=(0.5491*SalidaRed1[0]-0.3764*SalidaRed1[1]-
0.3570*SalidaRed1[2]+0.6857*SalidaRed1[3]+0.2062*SalidaRed1[4]-
```

```
0.1926*SalidaRed1[5]+0.2404*SalidaRed1[6]+0.4555*SalidaRed1[7]-
0.4699*SalidaRed1[8]-0.2170*SalidaRed1[9])-0.3439;


  NetaCapa2[7]=(0.1270*SalidaRed1[0]-
0.5735*SalidaRed1[1]+0.5082*SalidaRed1[2]+0.8516*SalidaRed1[3]+0.3744*Salid
aRed1[4]+0.4831*SalidaRed1[5]-0.0167*SalidaRed1[6]+0.2780*SalidaRed1[7]-
0.5524*SalidaRed1[8]-0.8624*SalidaRed1[9])-0.2452;


  NetaCapa2[8]=(0.1404*SalidaRed1[0]-0.8276*SalidaRed1[1]-
0.0045*SalidaRed1[2]+0.2007*SalidaRed1[3]+0.3910*SalidaRed1[4]+0.3323*Salid
aRed1[5]+0.6121*SalidaRed1[6]+0.1033*SalidaRed1[7]-0.5993*SalidaRed1[8]-
0.4064*SalidaRed1[9])-0.2358;


  NetaCapa2[9]=(-0.5783*SalidaRed1[0]-0.5750*SalidaRed1[1]-
0.7629*SalidaRed1[2]-0.7470*SalidaRed1[3]-0.0616*SalidaRed1[4]-
0.8676*SalidaRed1[5]+0.4753*SalidaRed1[6]+0.3699*SalidaRed1[7]-
0.2148*SalidaRed1[8]-0.1178*SalidaRed1[9])-0.0423;
```

Salida Segunda Capa Función de activación TanSig

```
  for(int j=0;j<=9;j=j+1){


    SalidaRed2[j]=NetaCapa2[j];


  }
```

Calculo de la Neta capa de salida de la red con pesos incluidos

```
  NetaCapaS[0]=(-0.4708*SalidaRed2[0]-0.7964*SalidaRed2[1]-
0.1151*SalidaRed2[2]-
0.1929*SalidaRed2[3]+0.0451*SalidaRed2[4]+0.1070*SalidaRed2[5]-
0.2552*SalidaRed2[6]+0.5636*SalidaRed2[7]-
0.7285*SalidaRed2[8]+0.3247*SalidaRed2[9])-0.7682;


  NetaCapaS[1]=(-
0.3966*SalidaRed2[0]+0.7789*SalidaRed2[1]+0.1777*SalidaRed2[2]+0.9096*Salid
aRed2[3]-
```

```
0.2798*SalidaRed2[4]+0.0627*SalidaRed2[5]+0.5421*SalidaRed2[6]+0.4333*Salid
aRed2[7]-0.5341*SalidaRed2[8]-0.5444*SalidaRed2[9])+0.4253;


  NetaCapaS[2]=(0.4116*SalidaRed2[0]+0.3365*SalidaRed2[1]-
0.4805*SalidaRed2[2]-0.0769*SalidaRed2[3]-0.1962*SalidaRed2[4]-
0.0699*SalidaRed2[5]-0.6151*SalidaRed2[6]+0.6728*SalidaRed2[7]-
0.3354*SalidaRed2[8]+0.3278*SalidaRed2[9])+0.0266;
```

Salida Ultima Capa Función de activación PureLin

```
  for(int k=0;k<=2;k=k+1){

    Salida[k]=NetaCapaS[k];

  }

  Serial.print(Salida[0]);

  Serial.print(Salida[1]);

  Serial.println(Salida[2]);

  delay(1000);
```

Color reconocido

```
  if(Salida[0]>Salida[1] && Salida[0]>Salida[2]){

    lcd.setCursor(1,0);

    lcd.print("      Rojo      ");

    digitalWrite(26,HIGH);
```

```
  digitalWrite(27,LOW);
  digitalWrite(28,LOW);
 }
 if(Salida[1]>Salida[0] && Salida[1]>Salida[2]){
  lcd.setCursor(1,0);
  lcd.print("      Verde     ");
  digitalWrite(26,LOW);
  digitalWrite(27,HIGH);
  digitalWrite(28,LOW);
 }
 if(Salida[2]>Salida[1] && Salida[2]>Salida[0]){
  lcd.setCursor(1,0);
  lcd.print("      Azul      ");
  digitalWrite(26,LOW);
  digitalWrite(27,LOW);
  digitalWrite(28,HIGH);
 }
```

```
  //delay(5000);

}
```

• **Resultados.** Vemos el circuito en funcionamiento de la herramienta emulando por medio de los cuatro *dip-switch* (de tres pines cada uno) los sensores de color TCS3200. Puesto que el arduino no tenía las suficientes entradas de puerto I^2C. Vemos que la LCD nos muestra el color que esta predominando, es decir, en la primera imagen el que predomina es el azul, en este caso los pines de posición no. 3, se encuentra activos (ON) de los *dip-switch* y se prendera el led azul. En la imagen dos el que predomina es el verde (los pines 2 de los *dip-switch*) se prendera el led verde y en la última imagen el que predomina es el rojo (los pines 1 de los *dip-switch*). Cuando todos los pines están en 0 se prendera el led azul por defecto esto nos indica que no hay ningún color que predomine.

Ilustración 40. Funcionamiento real de la herramienta (color azul)

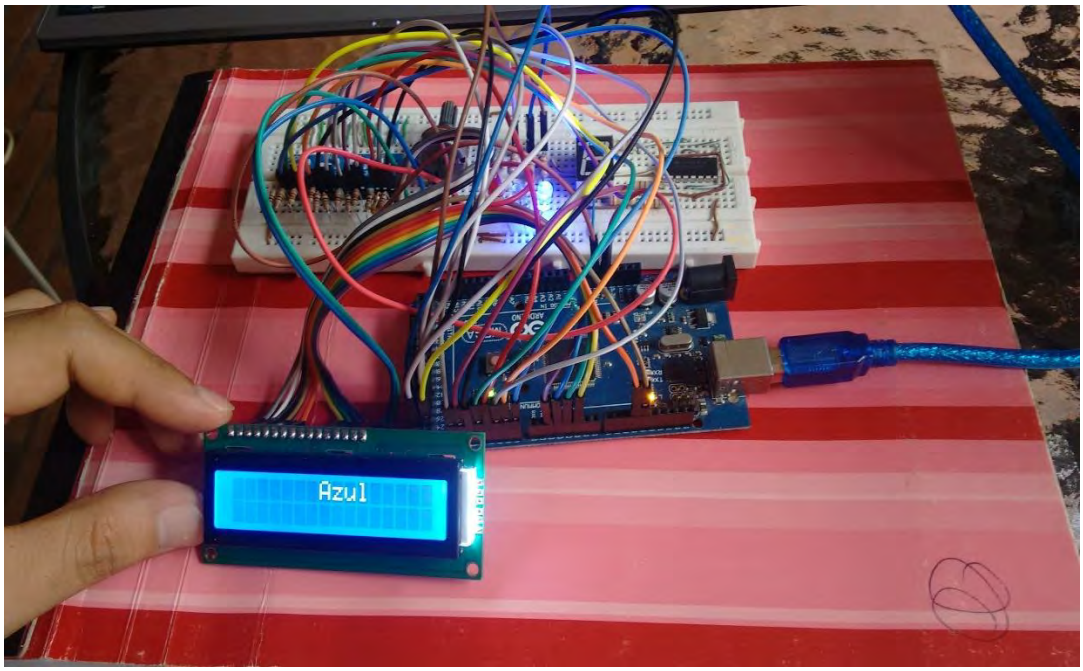


Ilustración 41. Funcionamiento real de la herramienta (color verde)

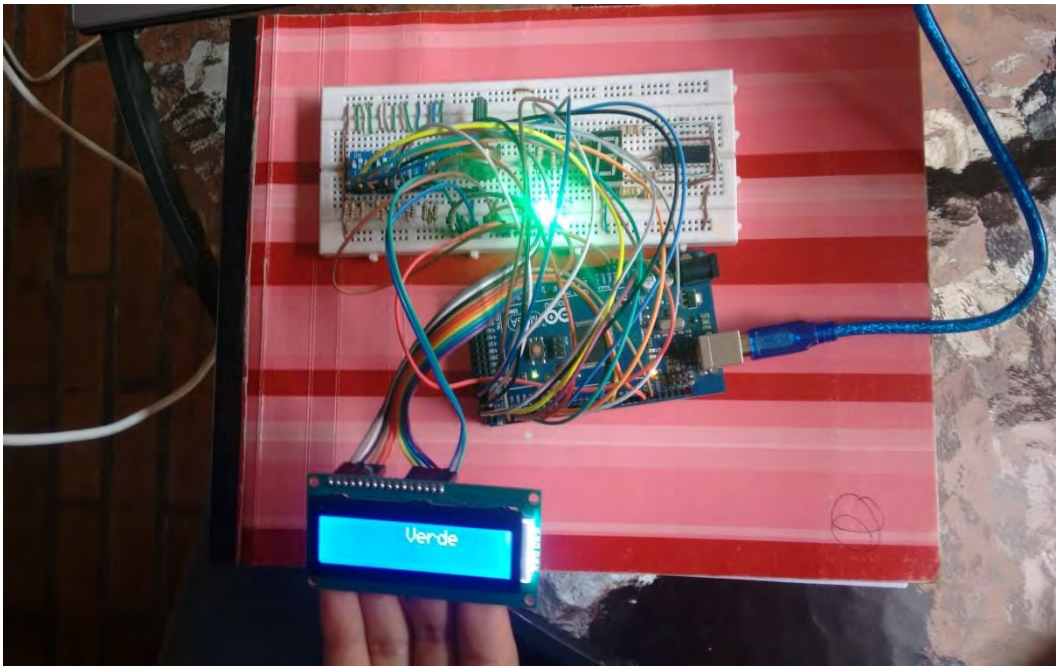
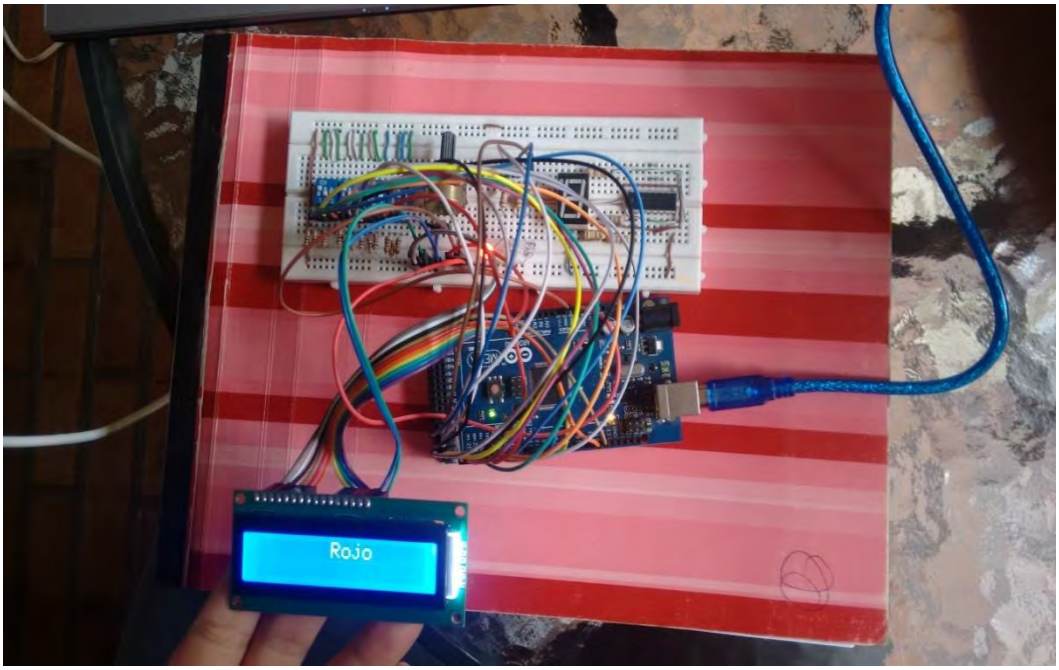


Ilustración 42. Funcionamiento real de la herramienta (color rojo)

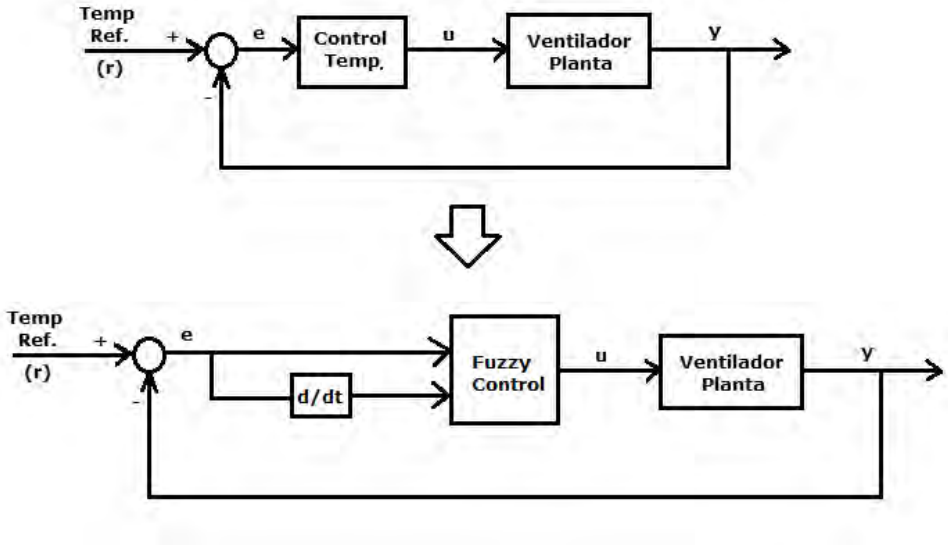


6.3 HERRAMIENTA No.3: CONTROL DIFUSO DE TEMPERATURA CON LM35

En esta primera implementación de fuzzy, el objetivo es mostrar por medio de una red neuronal de control fuzzy implementado con el sensor LM35 y un motor, el cual simula un ventilador. El cual según las lecturas obtenidas por el sensor LM35, el ciclo de motor aumenta o disminuye, de acuerdo a la escala de temperatura que se encuentre, para este caso el rango de trabajo del sensor es de 10°c a 60°c, por ejemplo, si el sensor está a temperatura ambiente (25°C), el ciclo de trabajo del motor es del 9%-18% aproximadamente, lo que significa que incrementa la velocidad del motor y si aumenta la temperatura hasta 60°C, el ciclo de trabajo del motor es alrededor del 30%- 40%, nos indica que aumenta casi llegando a su límite de velocidad.

Todo esto funciona con lógica difusa, del cual se crea un control inteligente utilizando las herramienta que brinda el toolbox Fuzzy Logic, de matlab, en los cuales se tienen los conjuntos difusos y sus respectivas funciones de pertenencia (frio, fresco, caluroso), cada uno tiene un rango de valores de los que incluye el conjunto y la salida es el ciclo de trabajo del pwm del motor el cual incluye las funciones de pertenencia(apagado, lento, medio, rápido) igual con los valores de los conjuntos con diferente valores del pwm del motor.

Ilustración 43. Diagrama de bloques del control de temperatura



En ilustración 42 nos muestra el toolbox del control difuso de temperatura, se muestra dos entradas ,el error, la derivada del error (Derror) y su respectiva

salida. En la ilustración 43 se muestra el error el cual se dividen en tres reglas que son caliente (funcion trapezoidal), neutral (funcion triangular) y frio(funcion trapezoidal), el cual esta entrada tiene un rango de temperatura de -50°c a 50°c. En la ilustración 44 se muestra la derivada del error (Derror) y sus respectivas reglas frio neutral y caliente.

Ilustración 44. Control fuzzy de temperatura con un lm35 en Matlab

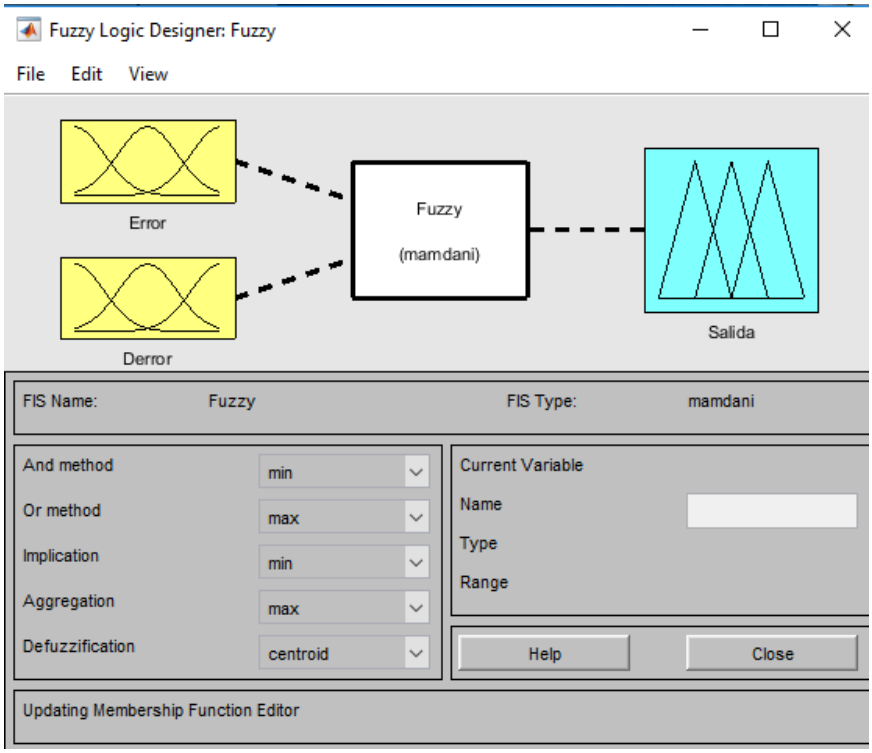


Ilustración 45. Función del error

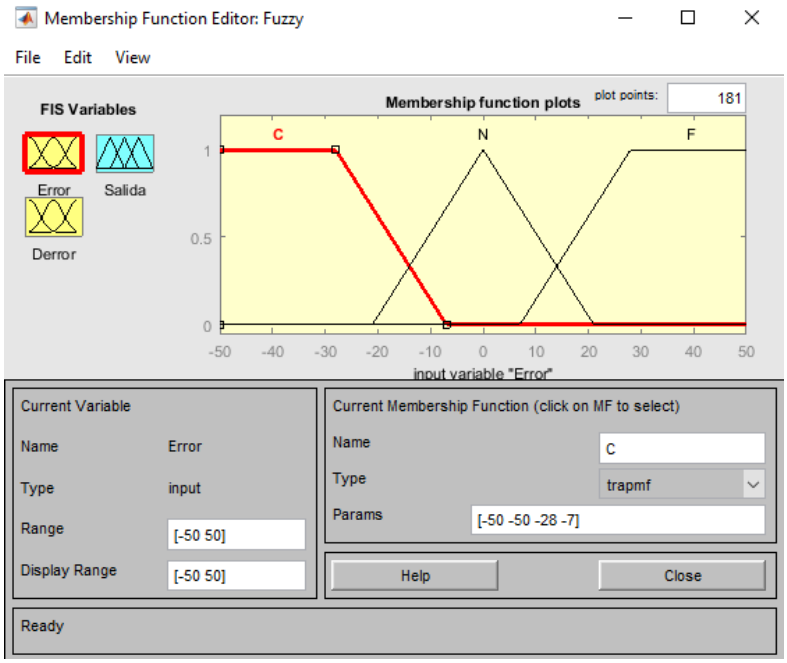
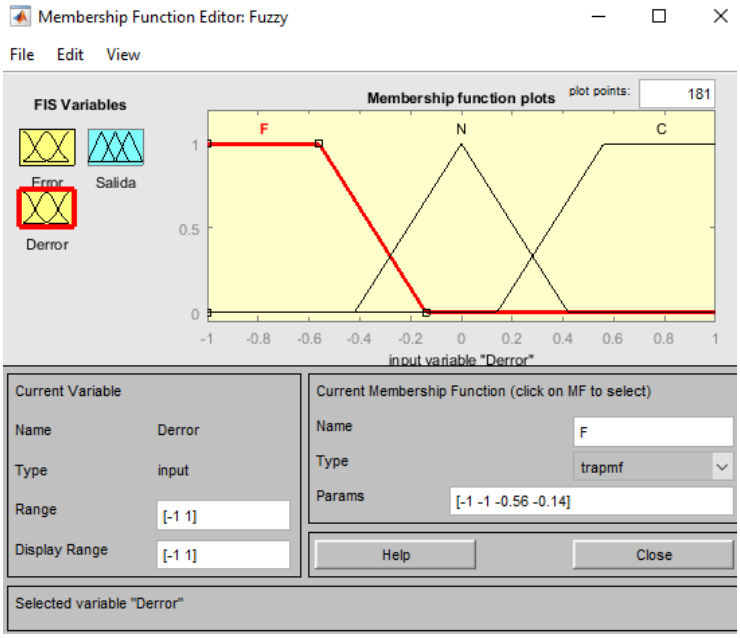


Ilustración 46. Función de la derivada del error.



En la ilustración 45 se muestra la salida del control difuso de temperatura, se implementara las reglas apagado, lento, promedio y rapido a un motor que hara la

tarea según la temperatura que marque el sensor LM35. El motor trabajara lento y se apagara si la temperatura es baja e ira rapido si la temperatura es alta.

Ilustración 47. Salida del sistema.

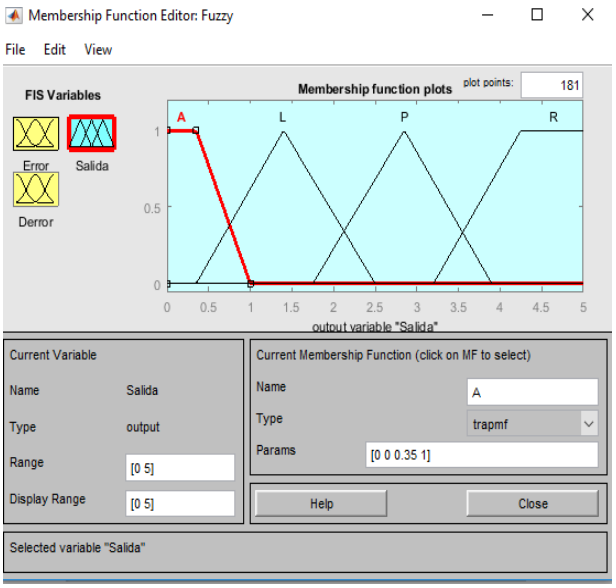


Tabla 3. Reglas lógicas

Reglas Lógicas:

$\frac{d}{dt}e(t)$ \ e	**C**	**N**	**F**
C	R	P	L
N	P	L	L
F	L	L	A

Donde:

A= Apagado

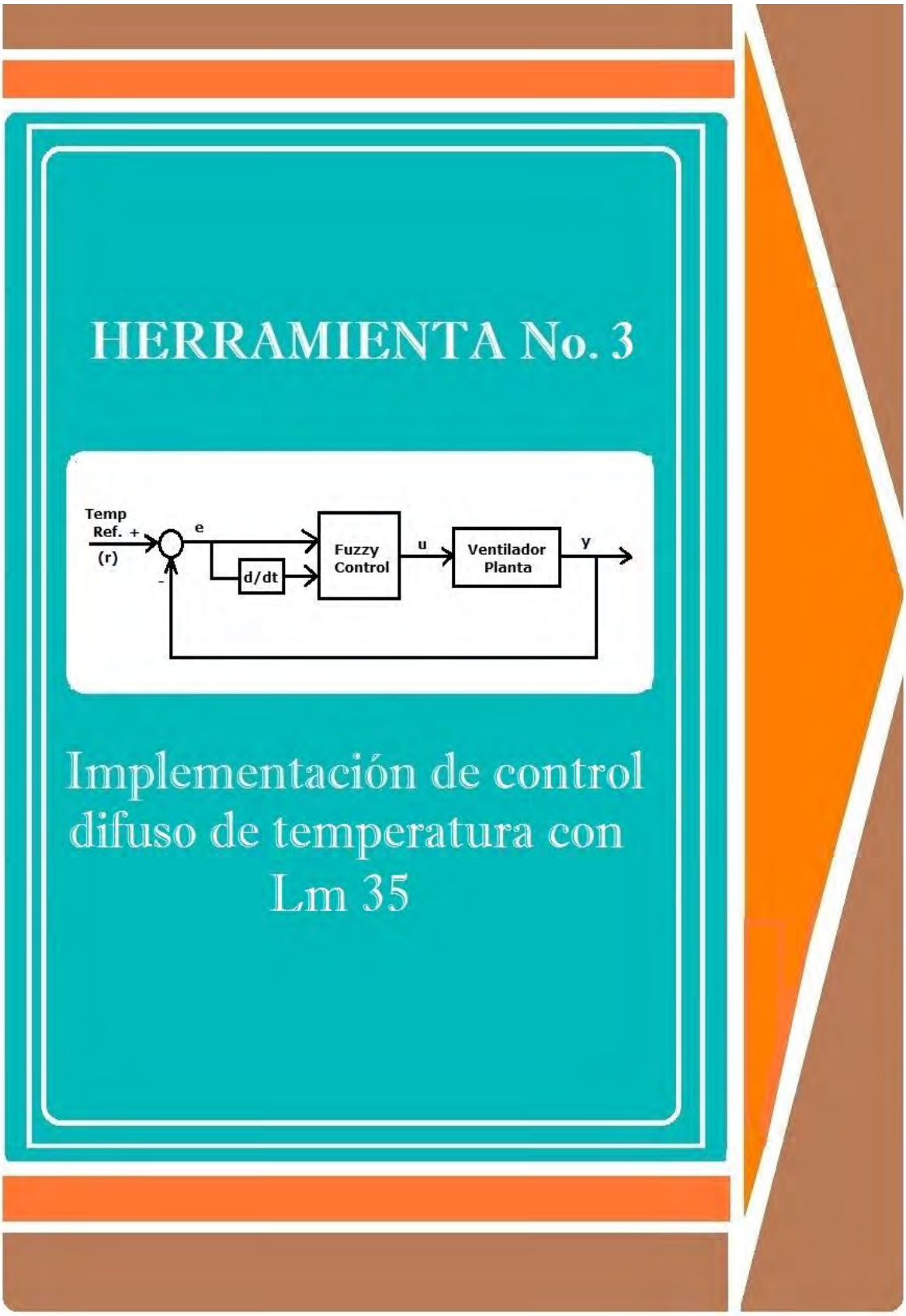
L= Lento

P= Promedio

R= Rápido

Ya conociendo previamente la descripción, se mostrará a continuación la guía de la herramienta No.3 del procedimiento realizado paso a paso:

Ilustración 48. Portada herramienta 3



- **Objetivos.** Diseñar un sistema de control difuso con Arduino.

Mostrar en detalle el proceso de implementación de un sistema de control difuso de temperatura con un sensor LM 35.

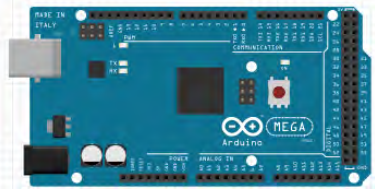
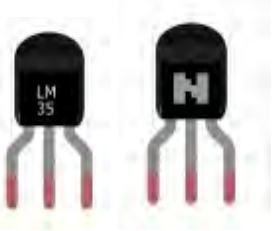
Implementar con Matlab las reglas de fuzificación para posteriormente realizar el procesamiento en Arduino.

- **Descripción del proyecto.** En esta herramienta consiste en realizar un control difuso, implementado con el sensor LM35 y un motor, el cual simula un ventilador. El cual según las lecturas obtenidas por el sensor LM35, el ciclo de motor aumenta o disminuye, de acuerdo a la escala de temperatura que se encuentre, para este caso el rango de trabajo del sensor es de 10°c a 60°c, por ejemplo, si el sensor está a temperatura ambiente (25°C), el ciclo de trabajo del motor es del 9%-18% aproximadamente, lo que significa que incrementa la velocidad del motor y si aumenta la temperatura hasta 60°C, el ciclo de trabajo del motor es alrededor del 30%- 40%, nos indica que aumenta casi llegando a su límite de velocidad.

Todo esto funciona con lógica difusa, del cual se crea un control inteligente utilizando las herramienta que brinda el toolbox Fuzzy Logic, de Matlab, en los cuales se tienen los conjuntos difusos y sus respectivas funciones de pertenencia (frio, fresco, caluroso), cada uno tiene un rango de valores de los que incluye el conjunto y la salida es el ciclo de trabajo del pwm del motor el cual incluye las funciones de pertenencia(apagado, lento, medio, rápido) igual con los valores de los conjuntos con diferente valores del pwm del motor.

- **Elementos requeridos**

Tabla 4. Materiales herramienta No.3

	ARDUINO MEGA
	LCD 16X2
	9 RESISTENCIAS
	UN SENSOR LM75 Y UN TRANSISTOR NPN
	2 POTENCIOMETROS DE 10KΩ
	2 AMPLIFICADORES OPERACIONAL
	UNA PROTOBOARD Y CABLES DE CONEXIÓN

• **Esquema electrónico del circuito.** En la ilustración 46 se muestra el esquema electrónico del montaje de la herramienta.

Ilustración 49. Esquema electrónico del control de temperatura.

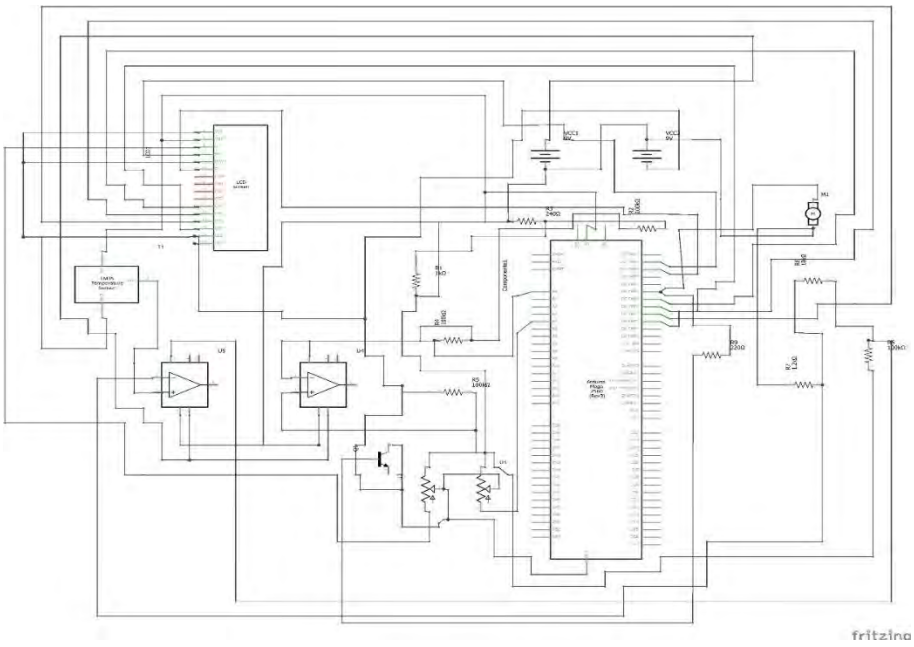
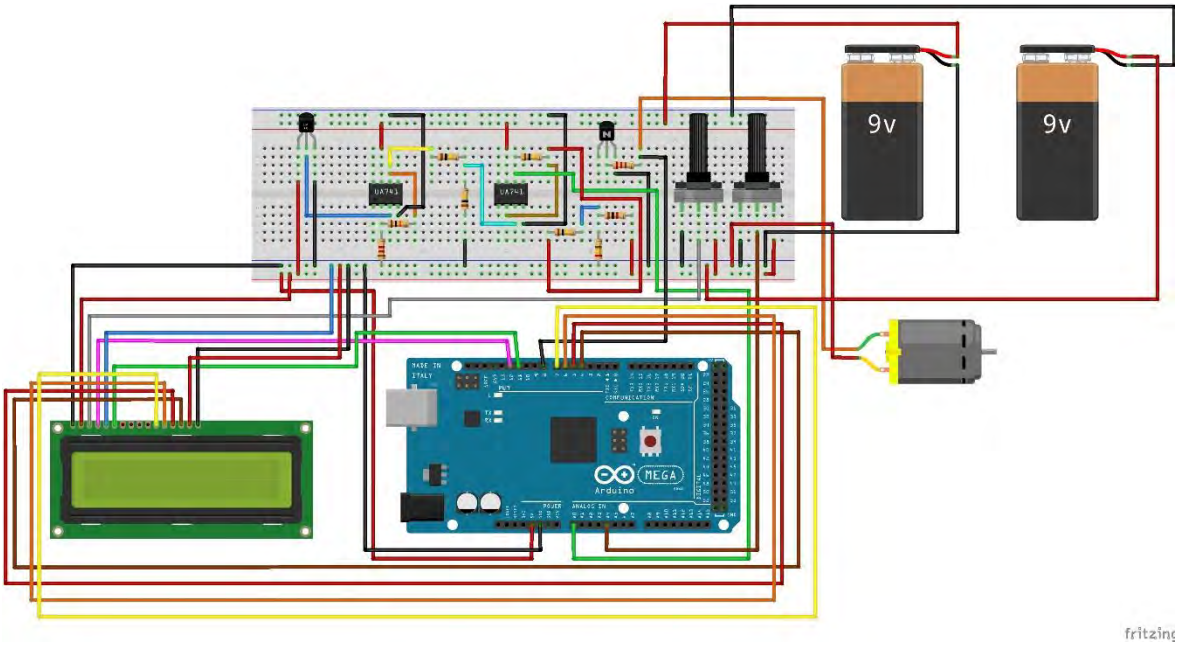


Ilustración 50. Conexiones con Arduino Mega y materiales requeridos.



- **Código. Este código esta implementado para un control difuso de temperatura con el sensor LM 35.**

Componentes de la librería fuzzy

```
#include <LiquidCrystal.h>

#include <FuzzyRule.h>

#include <FuzzyComposition.h>

#include <Fuzzy.h>

#include <FuzzyRuleConsequent.h>

#include <FuzzyOutput.h>

#include <FuzzyInput.h>
```

```
#include <FuzzyIO.h>

#include <FuzzySet.h>

#include <FuzzyRuleAntecedent.h>

int input = A0;

int soutput = 8;

int setPoint = A4;

int val=0;

int valSet=0;

double Error= 0;

LiquidCrystal lcd(12,11,7,6,5,4);
```

Paso 1: Iniciar de objeto de librería

```
 Fuzzy* fuzzy = new Fuzzy();

void setup(){

  lcd.begin(16,2);

  lcd.setCursor(0,0);

  lcd.print("Fuzzy Control");

  delay(3000);
```

```
 lcd.clear();

 pinMode(input,INPUT);

 pinMode(setPoint,INPUT);

 pinMode(soutput,OUTPUT);

Serial.begin(9600);
```

Paso 2: Crear una entrada fuzzy.

```
FuzzyInput* distance = new FuzzyInput(1);

FuzzySet* big = new FuzzySet(-50, -50, -28, -7);

distance->addFuzzySet(big);

FuzzySet* safe = new FuzzySet(-21, 0, 0, 21);

distance->addFuzzySet(safe);

FuzzySet* small = new FuzzySet(7, 28, 50, 50);

distance->addFuzzySet(small);

 fuzzy->addFuzzyInput(distance); // Add FuzzyInput to Fuzzy object
```

Paso 3: Creación de la velocidad de salida difusa

```
FuzzyOutput* velocity = new FuzzyOutput(1);// With its ID in param

FuzzySet* slow = new FuzzySet(0, 0, 0.5, 1);  Velocidad lenta
```

```
 velocity->addFuzzySet(slow); // Add FuzzySet

 FuzzySet* average = new FuzzySet(1, 2, 2, 3); Velocidad promedio

 velocity->addFuzzySet(average);

 FuzzySet* fast = new FuzzySet(3, 4, 4, 5); Velocidad rapida

 velocity->addFuzzySet(fast);

 fuzzy->addFuzzyOutput(velocity);

//Paso 4: Construir las reglas difusas

FuzzyRuleAntecedent* ifDistanceSmall = new FuzzyRuleAntecedent();

 ifDistanceSmall->joinSingle(small);

 FuzzyRuleConsequent* thenVelocitySlow = new FuzzyRuleConsequent();

 thenVelocitySlow->addOutput(slow);

FuzzyRule* fuzzyRule01 = new FuzzyRule(1, ifDistanceSmall, thenVelocitySlow);

 fuzzy->addFuzzyRule(fuzzyRule01);

FuzzyRuleAntecedent* ifDistanceSafe = new FuzzyRuleAntecedent();

 ifDistanceSafe->joinSingle(safe);

 FuzzyRuleConsequent* thenVelocityAverage = new FuzzyRuleConsequent();

 thenVelocityAverage->addOutput(average);
```

```
 FuzzyRule* fuzzyRule02 = new FuzzyRule(2, ifDistanceSafe, thenVelocityAverage);

   fuzzy->addFuzzyRule(fuzzyRule02);

  FuzzyRuleAntecedent* ifDistanceBig = new FuzzyRuleAntecedent();

 ifDistanceBig->joinSingle(big);

 FuzzyRuleConsequent* thenVelocityFast = new FuzzyRuleConsequent();

 thenVelocityFast->addOutput(fast);

 FuzzyRule* fuzzyRule03 = new FuzzyRule(3, ifDistanceBig, thenVelocityFast);

 fuzzy->addFuzzyRule(fuzzyRule03);

}

void loop(){

  val=analogRead(input);

  valSet=analogRead(setPoint);

  val=map(val,0,1023,10,60);

  valSet=map(valSet,0,1023,10,60);

  lcd.clear();

  lcd.setCursor(0,0);
```

```
  lcd.print("Sensor: ");
  lcd.setCursor(10,0);
  lcd.print(val);
  lcd.setCursor(0,1);
  lcd.print("SetPoint ");
  lcd.setCursor(11,1);
  lcd.print(valSet);
  Error = valSet - val;
```

Paso 5: Reporte del valor de entrada

```
fuzzy->setInput(1, Error);
```

Paso 6: Ejecutar fuzzyficacion

```
 fuzzy->fuzzify();
```

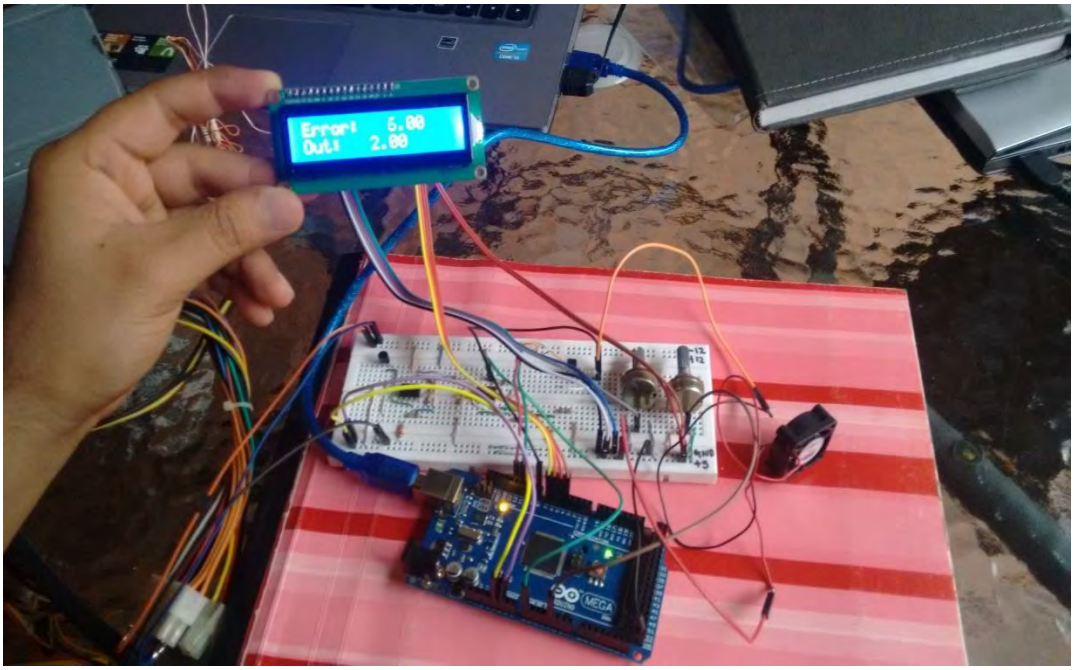
Paso 7: Ejecutar desifuzzificacion de cada entrada

```
 float output = fuzzy->defuzzify(1);
 lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Error: ");
```

```
  lcd.setCursor(9,0);

  lcd.print(Error);

  lcd.setCursor(0,1);

  lcd.print("Out: ");

  lcd.setCursor(7,1);

  lcd.print(output);

  output=map(output,0,5,0,255);

  analogWrite(soutput,output);

 delay(100);

}
```

- **Resultados.** Aquí se muestra la imagen de la herramienta en funcionamiento. Como se muestra la LCD muestra el valor del error y la salida del sensor LM35. El error es la diferencia que el sensor arroja respecto a la referencia que el potenciómetro de la derecha nos muestra. La salida es el voltaje que sale del sensor. Y el potenciómetro de la izquierda es ajustador de contraste de la LCD Cuando el error se encuentra en positivo como se muestra en la imagen quiere decir que la temperatura del sensor es baja y el motor va ir lento y entre más baje la temperatura se apagara. Cuando el error es negativo la temperatura en el sensor está subiendo y nuestro motor ira muy rápido.

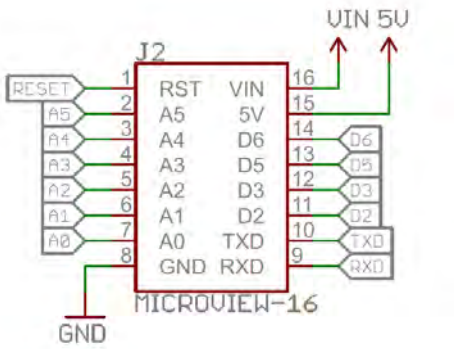
Ilustración 51. Funcionamiento real del control difuso de temperatura.



6.4 HERRAMIENTA No.4: VISUALIZACION DE UNA RED NEURONAL EN MICROVIEW (OLED)

Las redes neuronales son una implementación muy sencilla de un comportamiento local observado en nuestro cerebro. El cerebro está compuesto de neuronas, las cuales son elementos individuales de procesamiento. La información viaja entre las neuronas, y basado en la estructura y ganancia de los conectores neuronales, la red se comporta de forma diferente. A continuación, se muestra el esquemático del microview, cada una de sus entradas. Como se muestra en la ilustración 49.

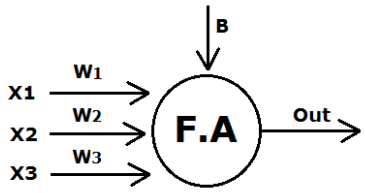
Ilustración 52. Esquema electrónico del microview.



Fuente: SPARKFUN, Start something, Arduino [en línea]. Colorado, 2012 [Consultado 10/03/2017]. Disponible en internet: https://cdn.sparkfun.com/datasheets/Dev/Arduino/.pdf

En este capítulo se simula por medio de un microview el comportamiento de la neurona ante variaciones de los pesos y la función de activación, reflejando en la pantalla del dispositivo la salida y la neta de la neurona. A continuación, se muestra en la ilustración 50 el esquema de la red neuronal con tres entradas implementada en la actividad:

Ilustración 53. Esquema de la red neuronal implementada en la actividad.



Donde W1, W2, W3 son los pesos asignados a la entrada de la red neuronal y B es el valor umbral.

La ecuación para calcular la neta está representada de la siguiente manera:

$$NETA = (W1X1 + W2X2 + W3X3) + B$$

La salida de la red dependerá de la función de activación de la neurona que están clasificadas en lineal o identidad, escalón o saturación y tansig.

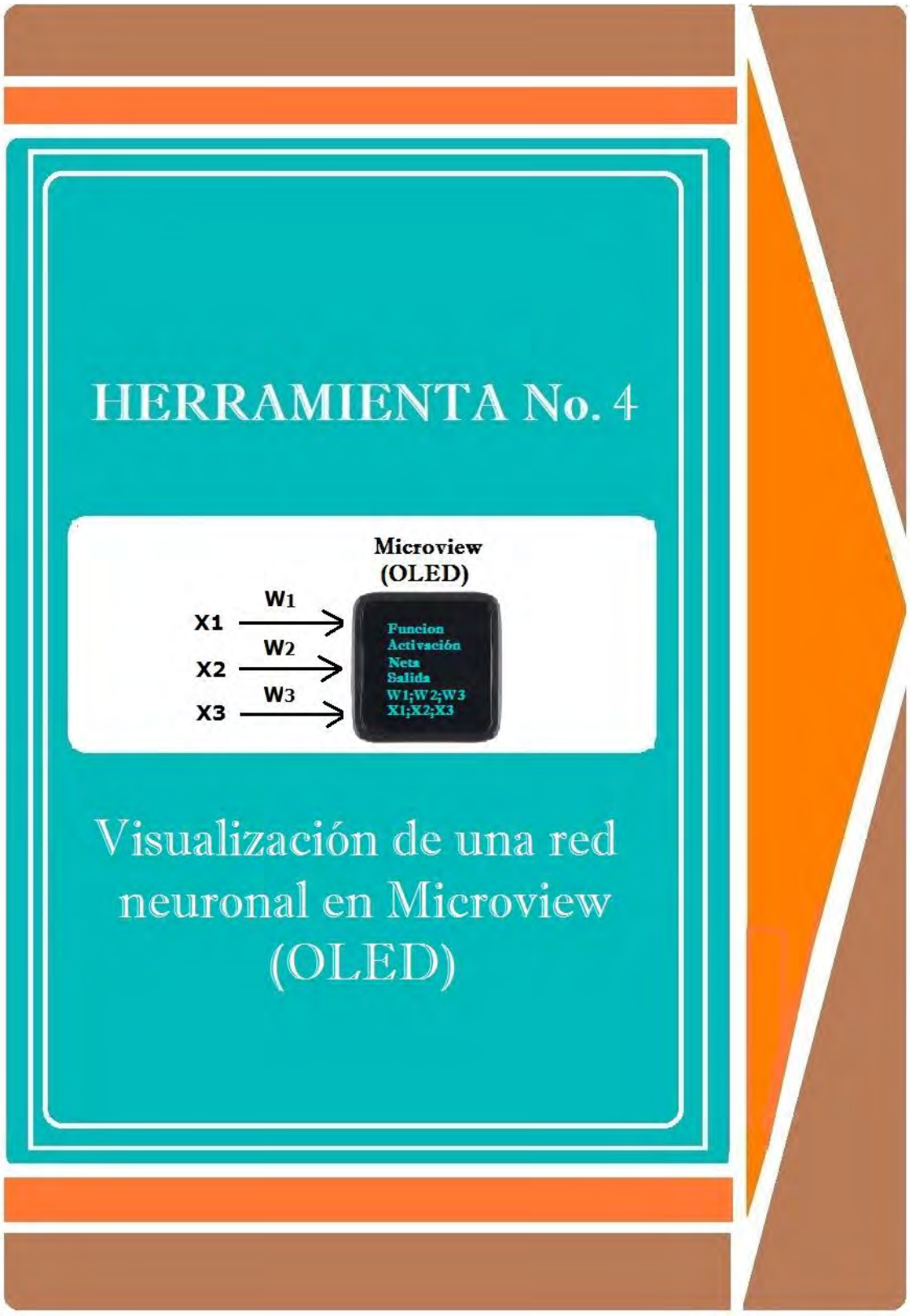
Para la codificación de la función de activación se hace uso de dos entradas digitales del microview como se muestra en la siguiente tabla 5.

Tabla 5. Funciones de activación de la red.

A(6)	B(A4)	OUT
0	0	No calcula
0	1	Lineal
1	0	Escalón
1	**1**	**Signoidal**

Ya conociendo previamente la descripción, se mostrará a continuación la guía de la herramienta No.4 del procedimiento realizado paso a paso:

Ilustración 54. Portada herramienta 4



• **Objetivos.** Conocer cómo se puede visualizar una red neuronal con sus respectivos parámetros.

Implementar una red neuronal en un módulo compatible con Arduino

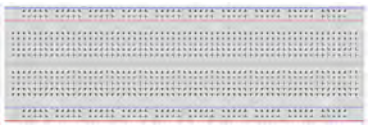
Utilizar librerías de Microview para la implementación gráfica de la red neuronal.

• **Descripción del proyecto.** Las redes neuronales son una implementación muy sencilla de un comportamiento local observado en nuestro cerebro. El cerebro está compuesto de neuronas, las cuales son elementos individuales de procesamiento. La información viaja entre las neuronas, y basado en la estructura y ganancia de los conectores neuronales, la red se comporta de forma diferente, En este capítulo se simula por medio de un microview el comportamiento de la neurona ante variaciones de los pesos y la función de activación, reflejando en la pantalla del dispositivo la salida y la neta de la neurona.

,

- **Materiales.**

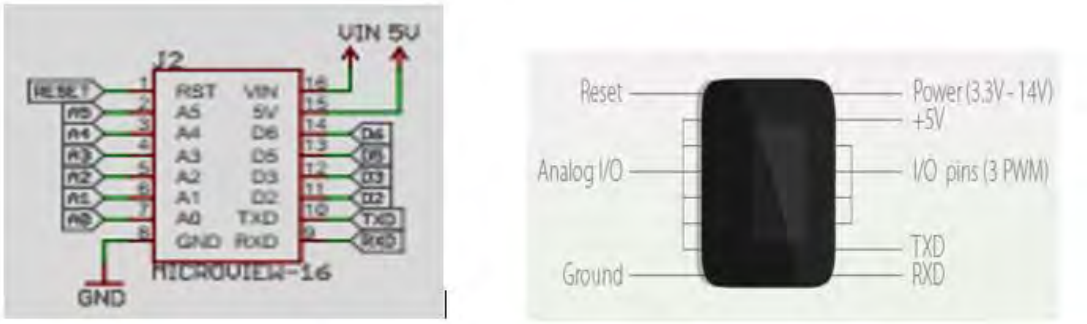
Tabla 6. Materiales Herramienta No. 4

	MICROVIEW
	5 RESISTENCIAS DE 220 Ω
	4 POTENCIOMETRO DE 10KΩ
	UN DIP SWITCH 8 SEGMENTOS
 	UNA PROTOBOARD Y CABLES DE CONEXIÓN

• **Esquema electrónico del circuito.** Vamos a montar un circuito que simule por medio de una microview una red neuronal con sus respectivos pesos, bias, la neta, su salida y por medio de un *dip-switch* sus respectivas entradas.

Para conectar la microview, se debe tener en cuenta el esquemático que se muestra la ilustración 51:

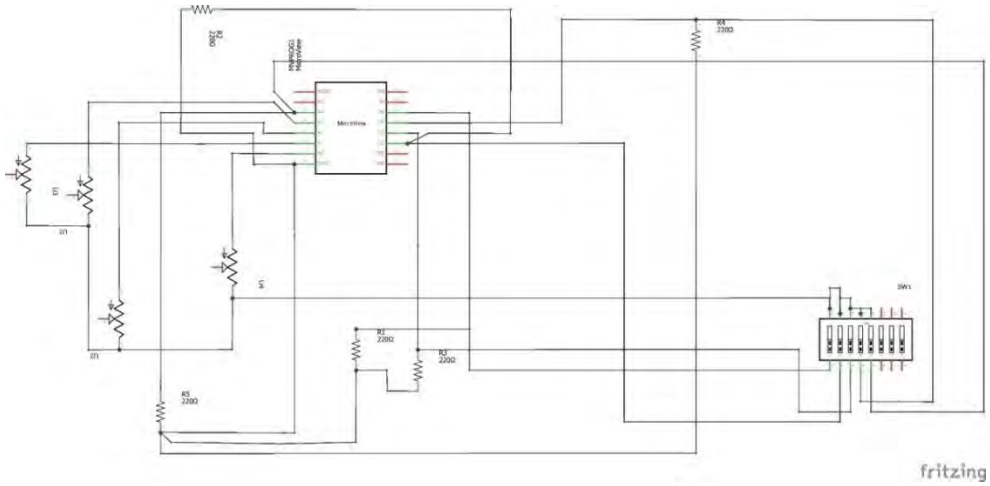
Ilustración 55. Esquemático de la Microview.



Fuente: SPARKFUN, Start something, Arduino [en línea]. Colorado, 2012 [Consultado 10/03/2017]. Disponible en internet: https://cdn.sparkfun.com/datasheets/Dev/Arduino/.pdf

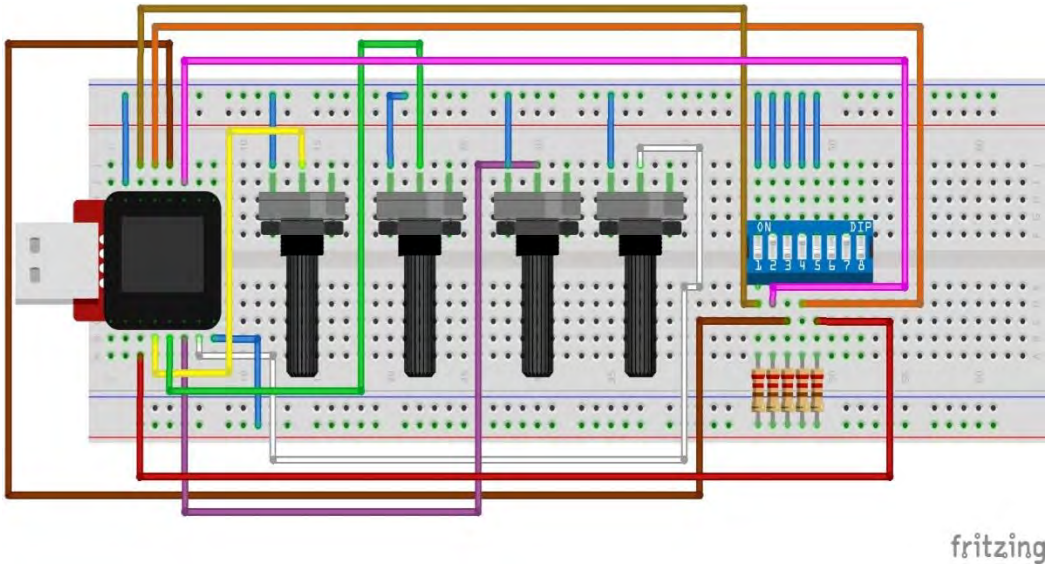
En la ilustración 52 se muestra el diagrama esquemático del circuito, que genera la herramienta Fritzing.

Ilustración 56. Diagrama esquemático.



Y el circuito en la protoboard, se puede visualizar por medio de la herramienta Fritzing, que nos facilita la visualización de cada una de las conexiones, como se muestra en la ilustración 53.

Ilustración 57. Conexiones con Arduino Mega y materiales requeridos.



- **Código.**

```
#include <MicroView.h>

Definir entradas

int X1=2;

int X2=3;

int X3=5;

//Definir Pesos

int W1=A0;
```

```
int W2=A1;

int W3=A2;
```

Definir Byas

```
int B=A3;
```

Definir Función Activación

```
int sel1=6;

int sel2=A4;
```

Definir Variables Globales

```
int in1;

int in2;

int in3;

float we1;

float we2;

float we3;

float by;

float Net;

float Salida;
```

```
void setup(){
  pinMode(X1,INPUT);
  pinMode(X2,INPUT);
  pinMode(X3,INPUT);
  pinMode(W1,INPUT);
  pinMode(W2,INPUT);
  pinMode(W3,INPUT);
  pinMode(B,INPUT);
  pinMode(sel1,INPUT);
  pinMode(sel2,INPUT);
  uView.begin();
  uView.clear(PAGE);
}
void loop(){
```

Lectura de Entradas

```
  in1=digitalRead(X1);
  in2=digitalRead(X2);
```

```
in3=digitalRead(X3);
```

Lectura de Pesos

```
we1=analogRead(W1);

we2=analogRead(W2);

we3=analogRead(W3);

by=analogRead(B);
```

Escalando Pesos

```
we1=Conversion(we1);

we2=Conversion(we2);

we3=Conversion(we3);

by=Conversion(by);
```

Calculo de Neta

```
Net=((in1*we1)+(we2*in2)+(we3*in3))+by;
```

Lectura de Selección

```
int a=digitalRead(sel1);

int b=digitalRead(sel2);

if(a==HIGH && b==HIGH){
```

Función Sigmoidal

```
  Salida=1/(1+exp(Net));
}
 if(a==LOW && b==HIGH){
  //Función Lineal
   Salida=Net;
 }
 if(a==HIGH && b==LOW){
   //Funcion Escalon
   if(Net>=0){
     Salida=1;
   }
   else{
     Salida=0;
   }
 }
 Entradas();
```

```
  uView.display();
  delay(2000);
  uView.clear(PAGE);
  Pesos();
  uView.display();
  delay(2000);
  uView.clear(PAGE);
  Neta();
  uView.display();
  delay(2000);
  uView.clear(PAGE);
  Output();
  uView.display();
  delay(2000);
  uView.clear(PAGE);
}
float Conversion(float we){
```

```
  we=(we/511)-1.0;
  return we;
}
void Entradas(){
  uView.setCursor(25,0);
  uView.print("In");
  uView.setCursor(0,7);
  uView.print("X1: "+String(in1));
  uView.setCursor(0,20);
  uView.print("X2: "+String(in2));
  uView.setCursor(0,33);
  uView.print("X3: "+String(in3));
}
void Pesos(){
  uView.setCursor(15,0);
  uView.print("Weights");
  uView.setCursor(0,7);
```

```
  uView.print("W1: "+String(we1));
  uView.setCursor(0,20);
  uView.print("W2: "+String(we2));
  uView.setCursor(0,33);
  uView.print("W3: "+String(we3));
}
void Neta(){
  uView.setCursor(20,0);
  uView.print("NETA");
  uView.setCursor(0,20);
  uView.print("Neta: "+String(Net));
}
void Output(){
  uView.setCursor(25,0);
  uView.print("OUT");
  uView.setCursor(0,20);
  uView.print("Sal: "+String(Salida)); }
```

• **Resultados.** Aquí se ve el montaje de la herramienta con la microview. Se está simulando una red neuronal con sus respectivas entradas, pesos, la neta y la salida. En el *dip- switch* se activa las entradas digitales, X1 es el pin 2, X2 es el pin 3 y X3 es el pin 4. Los pines 1 y 5 son las entradas A y B para poder definir que función de activación está implementando la red. La primera imagen muestra las entradas, en la segunda los pesos, en la tercera la neta y en la cuarta la salida.

Ilustración 58. Funcionamiento real de la herramienta (se visualiza las entradas).

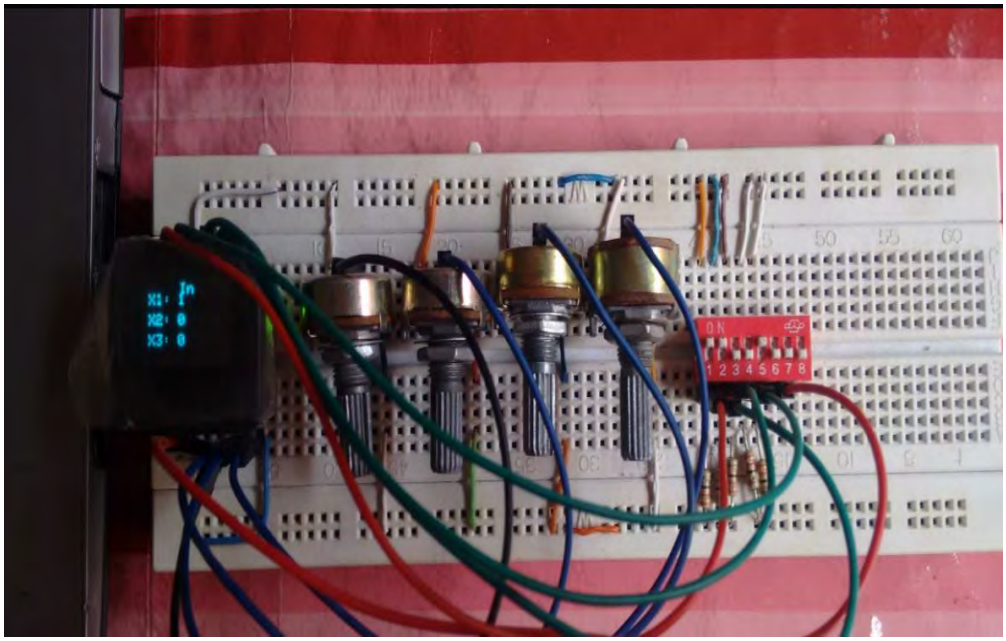


Ilustración 59. Funcionamiento real de la herramienta (se visualiza los pesos).

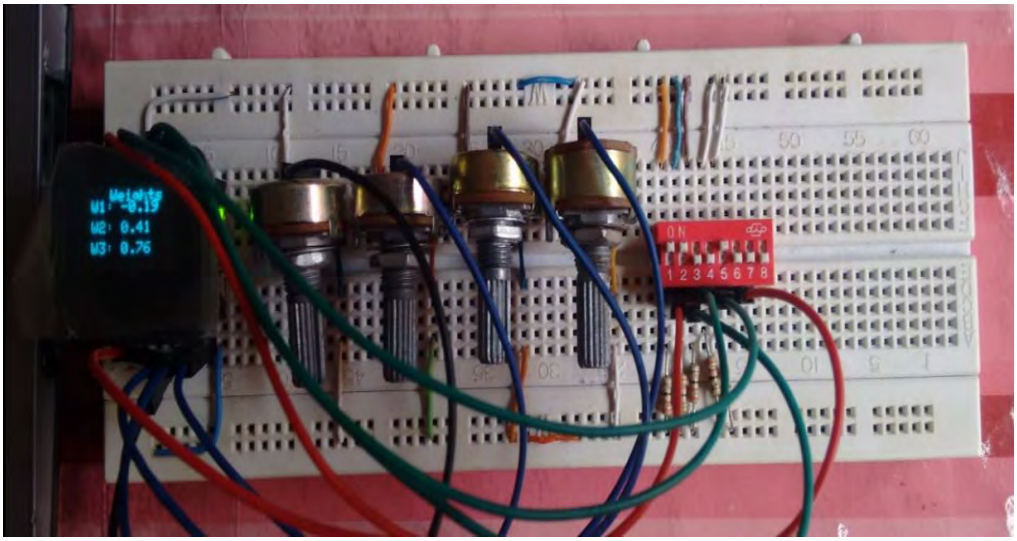


Ilustración 60. Funcionamiento real de la herramienta (se visualiza la neta).

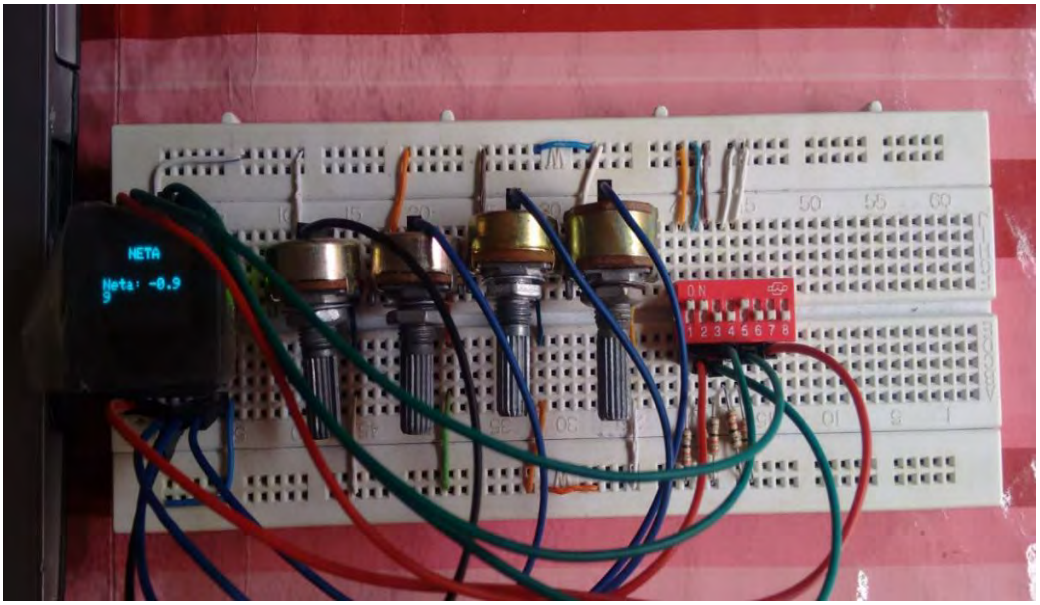
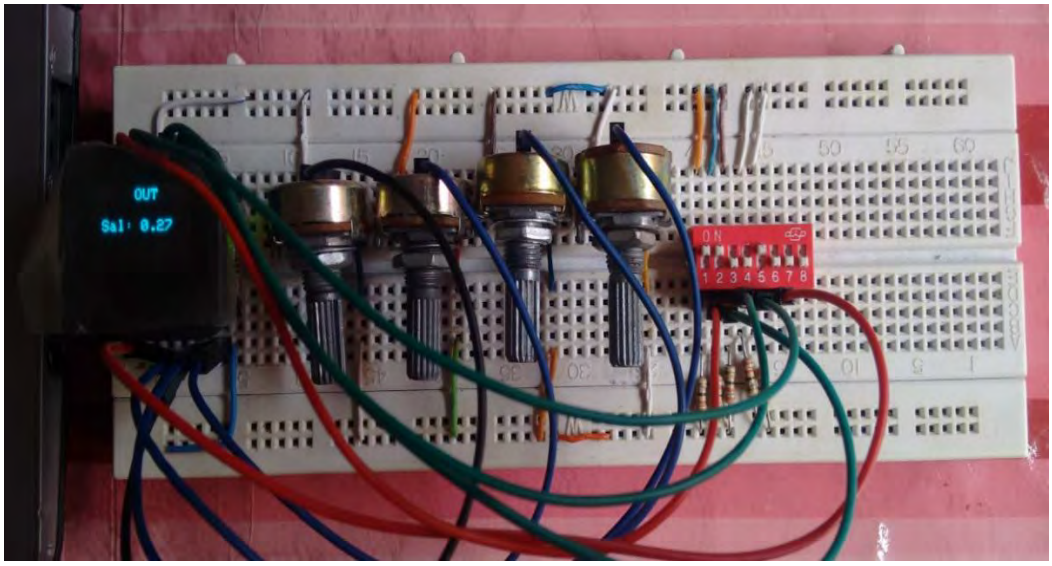


Ilustración 61. Funcionamiento real de la herramienta (se visualiza la salida).



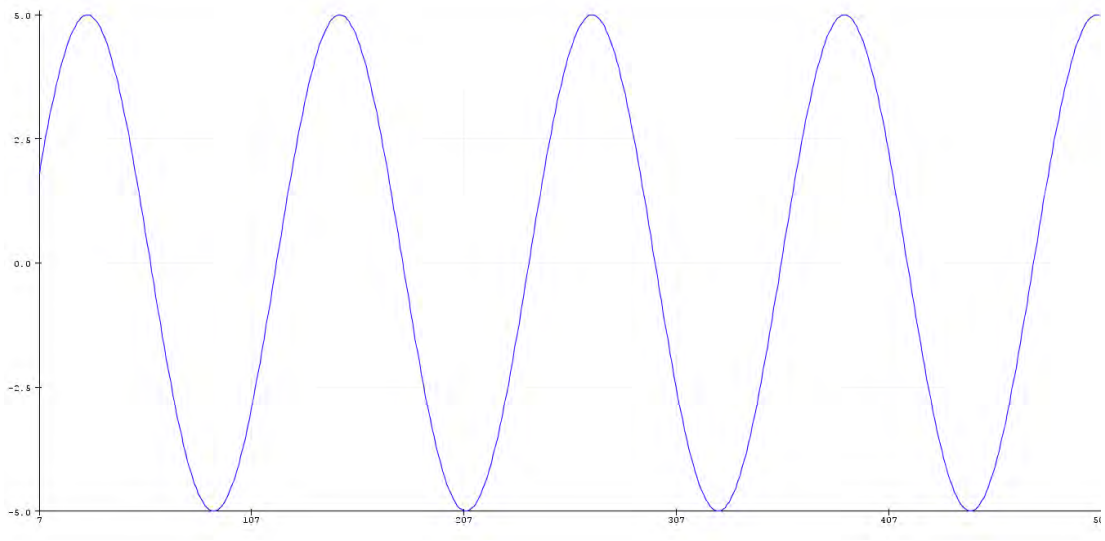
6.5 HERRAMIENTA No.5: VISUALIZACION DE SEÑALES SENO, COSENO Y EMULACIÓN DE UNA PLANTA DE PRIMER ORDEN, EN MATLAB Y ARDUINO

En esta herramienta vamos a simular por medio del Tollbox de Matlab, la emulación de las señales seno y la emulación de una planta de primer orden. Las redes neuronales son una implementación muy sencilla de un comportamiento local observado en nuestro cerebro.

Se implementa el entrenamiento de una red neuronal en arduino de las funciones seno, seno con coseno y el funcionamiento de una planta de primer orden. Se entrenará la red neuronal primero en Matlab (se anexará el código más adelante) y posteriormente se introducirá en el arduino para que este emule la funciones.

6.5.1 Función seno.

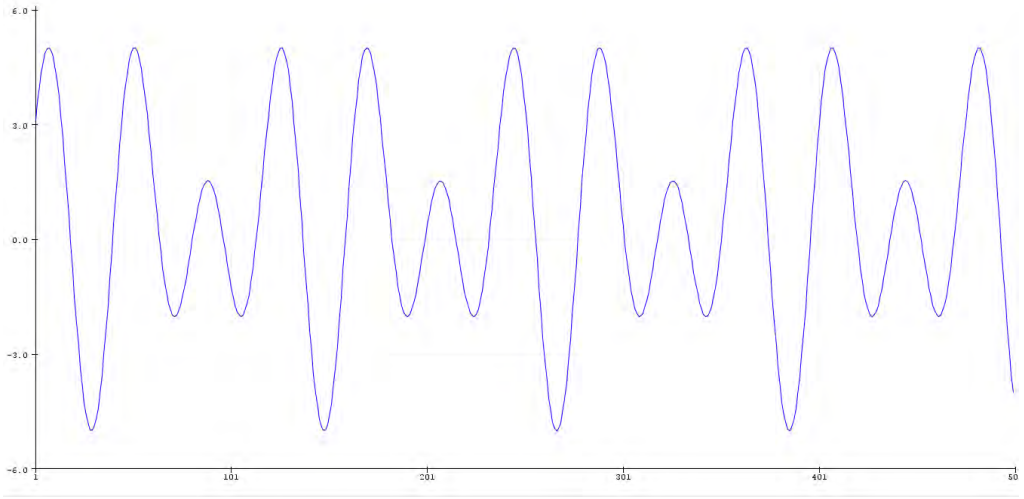
Ilustración 62. Función seno.



Mostramos la función seno graficada en arduino con la red neuronal entrenada previamente en Matlab y posteriormente introducida en arduino. Para poder graficar esta función en arduino primero subimos el código a arduino,

6.5.2 Función seno con coseno.

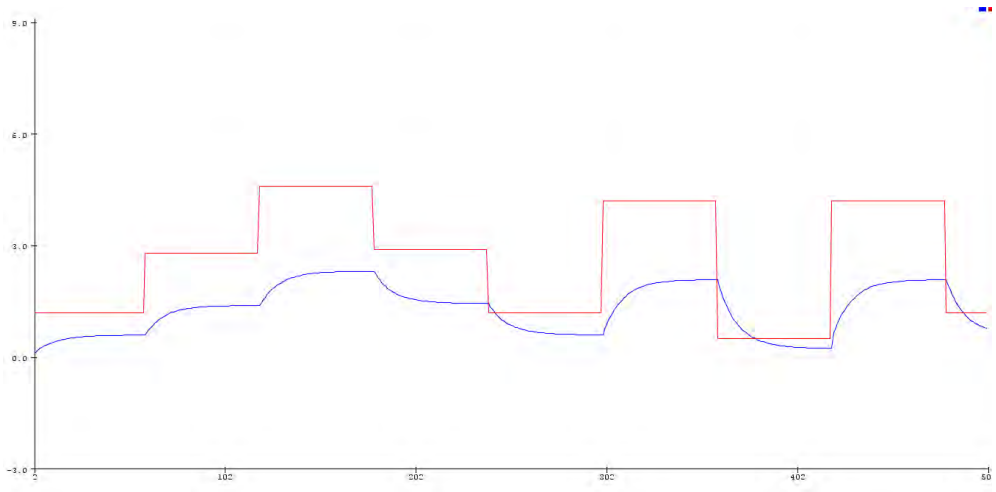
Ilustración 63. Función seno con coseno.



Se muestra la ilustración 59 la función seno con coseno entrenada la red con arduino.

6.5.3 Planta de primer orden. En la ilustración 60 Se muestra la gráfica de la planta de primer orden entrenada la red con Arduino.

Ilustración 64. Planta de primer orden.



Tomas de datos planta de primer orden en simulink:

Ilustración 65. Planta simulada en simulink.

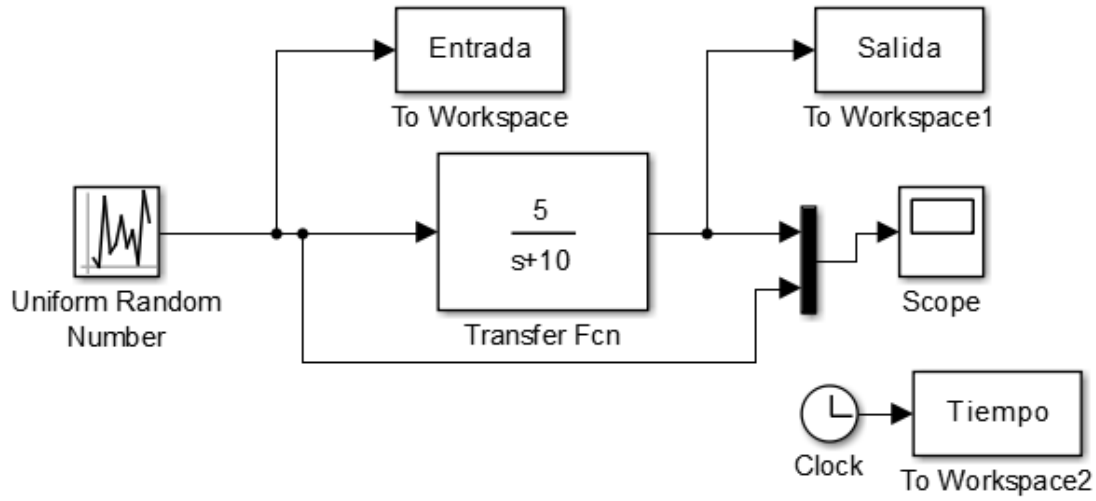
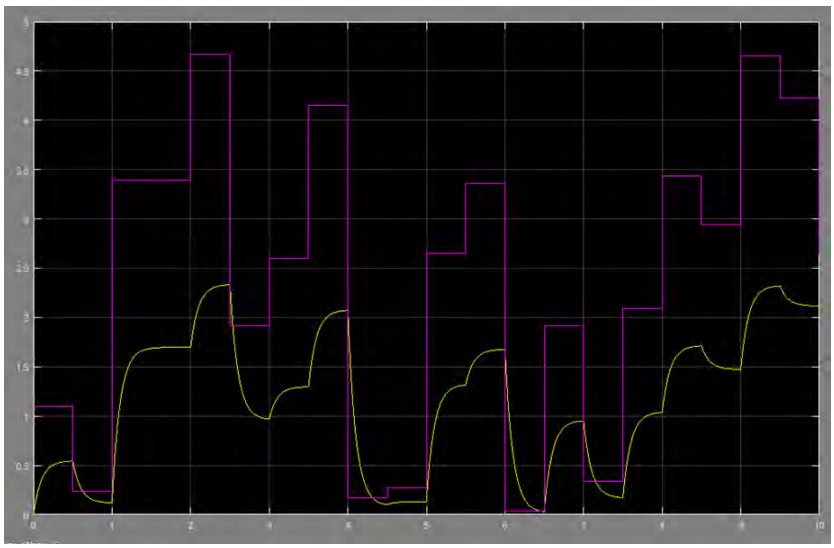
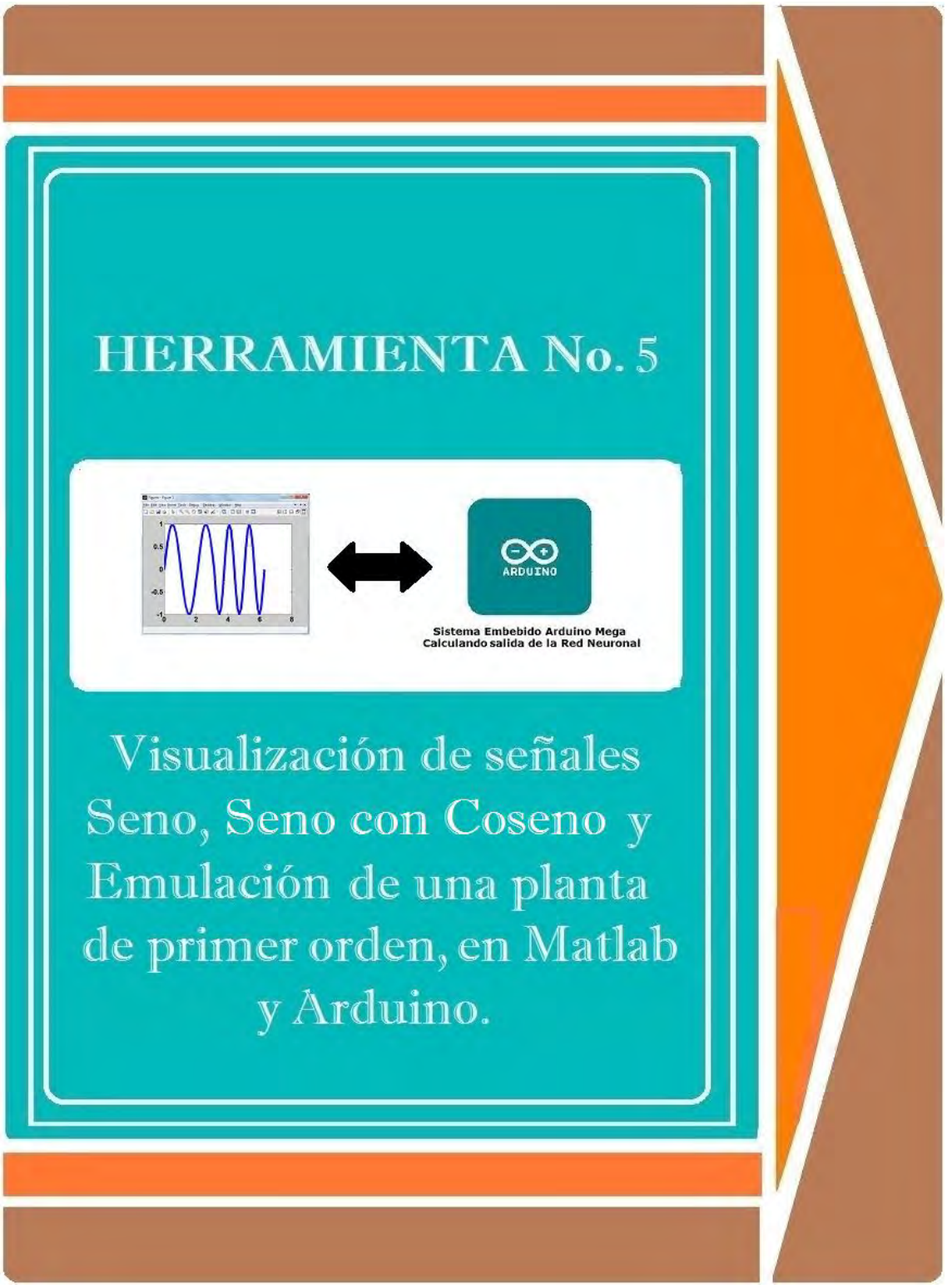


Ilustración 66. Resultado de la simulación en simulink.



Ya conociendo previamente la descripción, se mostrará a continuación la guía de la herramienta No.5 del procedimiento realizado paso a paso:

Ilustración 67. Portada herramienta 4



- **Objetivos.** Implementar las funciones seno, seno con coseno y una planta de primer orden utilizando Arduino y Matlab.

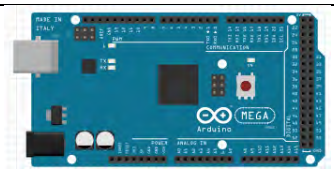
Mostrar el procesamiento de una red neuronal con arduino.

Realizar la validación de una red neuronal en Matlab.

- **Descripción del proyecto.** Se implementa el entrenamiento de una red neuronal en arduino de las funciones seno, seno con coseno y el entrenamiento de una planta de primer orden. Se entrenará la red neuronal primero en Matlab (se anexará el código más adelante) y posteriormente se codificará en el arduino para que este emule la funciones.

- **Materiales requeridos.**

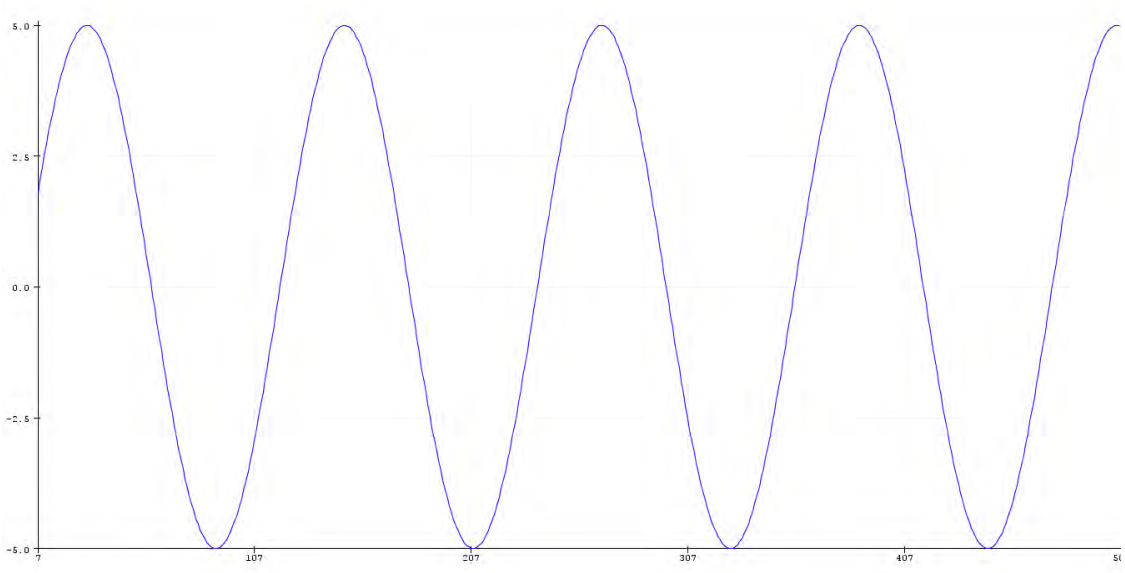
Tabla 7. Materiales herramienta 5

	ARDUINO MEGA

- **Código y resultados.**

Función seno.

Ilustración 68. Resultado en arduino de la función seno.

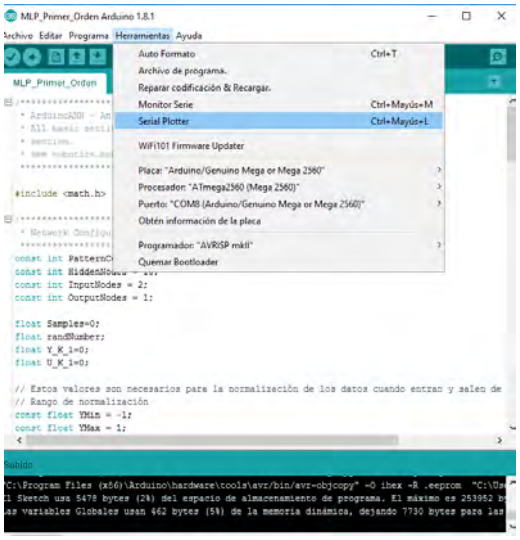


En la ilustración 63 se muestra la función seno graficada en Arduino con la red neuronal entrenada previamente en Matlab y posteriormente introducida en Arduino. Para poder graficar esta función en Arduino primero subimos el código.

Ilustración 69. Subiendo el programa al Arduino.

Segundo vamos a herramientas y damos click en Seria Plotter y obtenemos la gráfica de seno.

Ilustración 70. Visualización del serial plotter.



Todos estos pasos se deben realizar en las demás funciones que vamos a simular y así obtener sus respectivas gráficas.

- **Código.**

```
#include <math.h>

const int PatternCount = 1;

const int HiddenNodes = 10;

const int InputNodes = 1;

const int OutputNodes = 1;
```

Estos valores son necesarios para la normalización de los datos cuando entran y salen de la red

Rango de normalización

```
const float YMin = -1;

const float YMax = 1;
```

Valores para normalizar la entrada de la red

```
const float XMin = 0;

const float XMax = 6.2832;
```

Valores para normalizar la salida de la red

```
const float DMin = -5;

const float DMax = 5;
```

Los pesos de la red fueron obtenidos en Matlab y se copiaron a este programa

Pesos capa oculta

```
const float HiddenWeights[HiddenNodes][InputNodes+1]= {

{   12.1008 , -12.2479},

{   -2.0963 ,   1.9875},

{   -6.6196 ,   4.2421},

{   -6.2267 ,   2.1509},

{    2.1891 ,  -0.2349},

{    2.9252 ,   0.5600},

{    6.3450 ,   2.2821},

{   -2.9795 ,  -2.3524},

{    5.3267 ,   5.0348},

{   8.6169  ,  9.1310}

 };
```

Pesos capa de salida

```
const float OutputWeights[OutputNodes][HiddenNodes+1]  = {
```

```
 {0.0046,  -1.5960,  -0.0046,  0.0045,  -1.3084, -0.3446,  -0.0055,  -0.7463,
0.1888,  0.1186, 0.3803 }

};

 int i, j, p, q, r;

float Accum;

float Hidden[HiddenNodes];

float Output[OutputNodes];

float Input[InputNodes][PatternCount];

 void setup(){

 Inicio conexión serial

  Serial.begin(9600);

}

void loop(){

   float Entrada;

  float Salida;

  float Tiempo;

Tiempo=millis();
```

```
Entrada=(fmod(Tiempo,6283))/1000;
```

Normalización de la entrada que se usará en la red neuronal

```
Input[0][0]=YMin+ ((Entrada-XMin)*((YMax-YMin)/(XMax-XMin)));
```

/*

* Calculo de la activación de la capa oculta**

***/**

```
   for( i = 0 ; i < HiddenNodes ; i++ ) {
     Accum = HiddenWeights[i][InputNodes] ;
     for( j = 0 ; j < InputNodes ; j++ ) {
       Accum += HiddenWeights[i][j]*Input[j][0];
     }
     Hidden[i] = (2.0/(1.0 + exp(-2*Accum)))-1.0;
   }
```

```
/*****************************************************************
```

*** Calculo de la activación de la capa de salida**

```
*****************************************************************/

   for( i = 0 ; i < OutputNodes ; i++ ) {

     Accum = OutputWeights[i][HiddenNodes] ;

     for( j = 0 ; j < HiddenNodes ; j++ ) {

       Accum +=  OutputWeights[i][j]*Hidden[j];

     }

  Output[i] = Accum;

   }
```

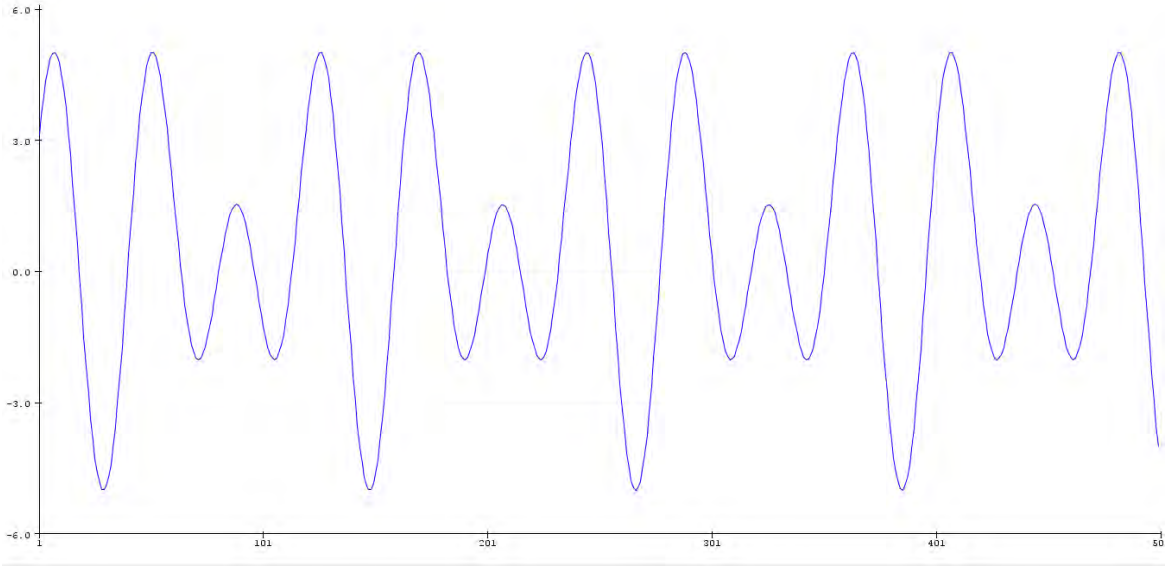
La salida da la red está normalizada, para que quede en el rango original hay que desnormalizar

Desnormalización de la salida de la red neuronal

```
Salida=DMin+ ((Output[0]-YMin)*((DMax-DMin)/(YMax-YMin)));

Serial.println( Salida);       // print as an ASCII-encoded decimal - same as "DEC"

delay(50);

}
```

Función seno con coseno.

Ilustración 71. Función seno con coseno.



En la ilustración 66 se muestra la función seno con coseno entrenada la red con Arduino.

- **Código.**

```
#include <math.h>

/******************************************************************

 * Conexión de la red personalizada

 ******************************************************************/

const int PatternCount = 1;

const int HiddenNodes = 10;
```

```
const int InputNodes = 1;

const int OutputNodes = 1;
```

Estos valores son necesarios para la normalización de los datos cuando entran y salen de la red

Rango de normalización

```
const float YMin = -1;

const float YMax = 1;
```

Valores para normalizar la entrada de la red

```
const float XMin = 0;

const float XMax = 6.2832;
```

Valores para normalizar la salida de la red

```
const float DMin = -5;

const float DMax = 5;
```

Los pesos de la red fueron obtenidos en Matlab y se copiaron a este programa

Pesos capa oculta

```
const float HiddenWeights[HiddenNodes][InputNodes+1]= {

 { -5.6372,    5.3821},
```

{ 10.4686, -7.6237},

{ 5.1070, -3.2261},

{ 4.4430, -1.4989},

{ 4.3611, -0.1144},

{ 8.6242,   0.7900},

{ -4.1019, -1.3459},

{ 4.4553,   2.9074},

{ -9.3769, -7.1024},

{ 5.4661,   5.4731}

};

Pesos capa de salida

const float OutputWeights[OutputNodes][HiddenNodes+1] = {

{-1.2589, -0.0347, -1.1682, 1.5808, -2.3183, -0.0512, -3.0535, -2.6312, 0.0598, 1.4445, 0.3498}

};

int i, j, p, q, r;

```
float Accum;

float Hidden[HiddenNodes];

float Output[OutputNodes];

float Input[InputNodes][PatternCount];

void setup(){
```

Inicio de la conexión serial

```
  Serial.begin(9600);

}

void loop(){

  float Entrada;

  float Salida;

  float Tiempo;

Tiempo=millis();

Entrada=(fmod(Tiempo,6283))/1000;
```

Normalización de la entrada que se usará en la red neuronal

```
Input[0][0]=YMin+ ((Entrada-XMin)*((YMax-YMin)/(XMax-XMin)));
```

```
/*****************************************************************
* Calculo de las activaciones de la capa oculta
*****************************************************************/
   for( i = 0 ; i < HiddenNodes ; i++ ) {
     Accum = HiddenWeights[i][InputNodes] ;
     for( j = 0 ; j < InputNodes ; j++ ) {
       Accum += HiddenWeights[i][j]*Input[j][0];
     }
     Hidden[i] = (2.0/(1.0 + exp(-2*Accum)))-1.0;
   }
/*****************************************************************
* Calculo de las activaciones de la capa de salida
*****************************************************************/
   for( i = 0 ; i < OutputNodes ; i++ ) {
     Accum = OutputWeights[i][HiddenNodes] ;
     for( j = 0 ; j < HiddenNodes ; j++ ) {
       Accum +=  OutputWeights[i][j]*Hidden[j];
```

```
    }

  Output[i] = Accum;

  }
```

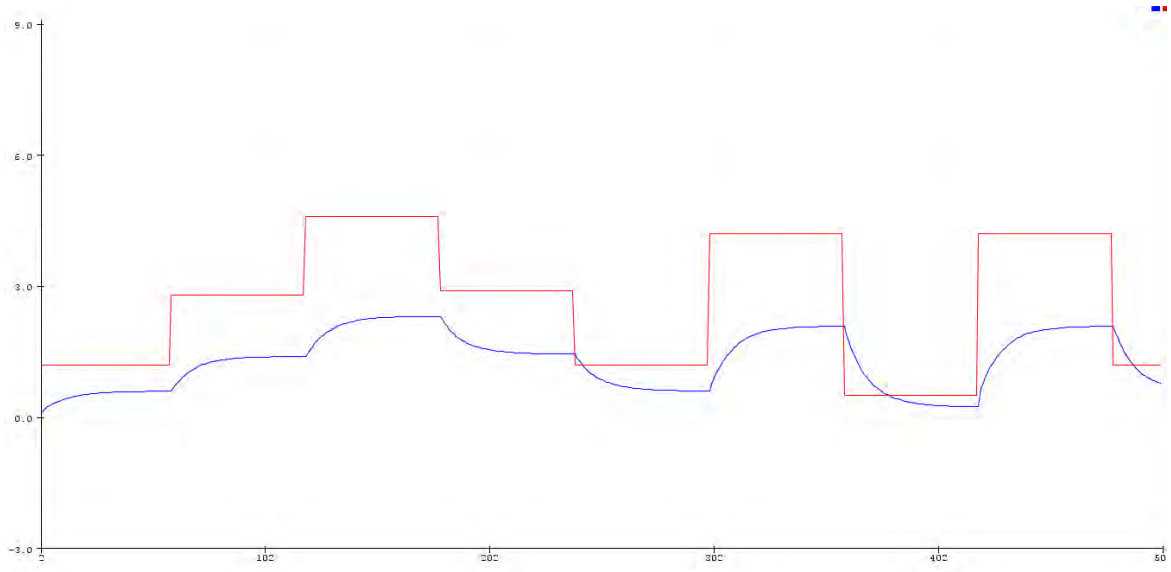
La salida da la red está normalizada, para que quede en el rango original hay que desnormalizar

Desnormalización de la salida de la red neuronal

```
Salida=DMin+ ((Output[0]-YMin)*((DMax-DMin)/(YMax-YMin)));

Serial.println( Salida);       // print as an ASCII-encoded decimal - same as "DEC"

delay(50);  }
```

Planta de primer orden.

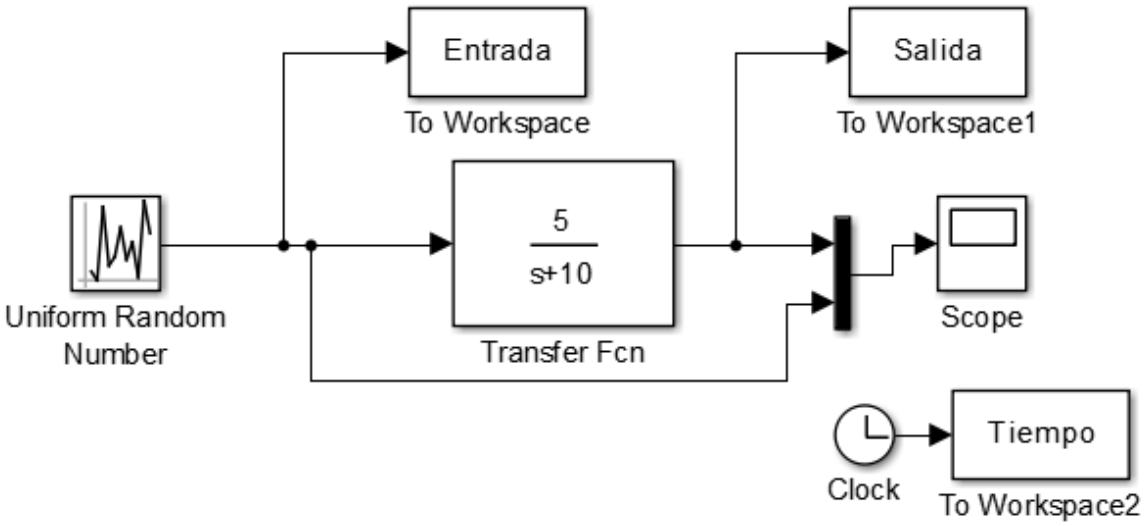
Ilustración 72. Resultado de la planta de primer orden.



Se muestra la Ilustración 67 la planta de primer orden entrenada la red con Arduino.

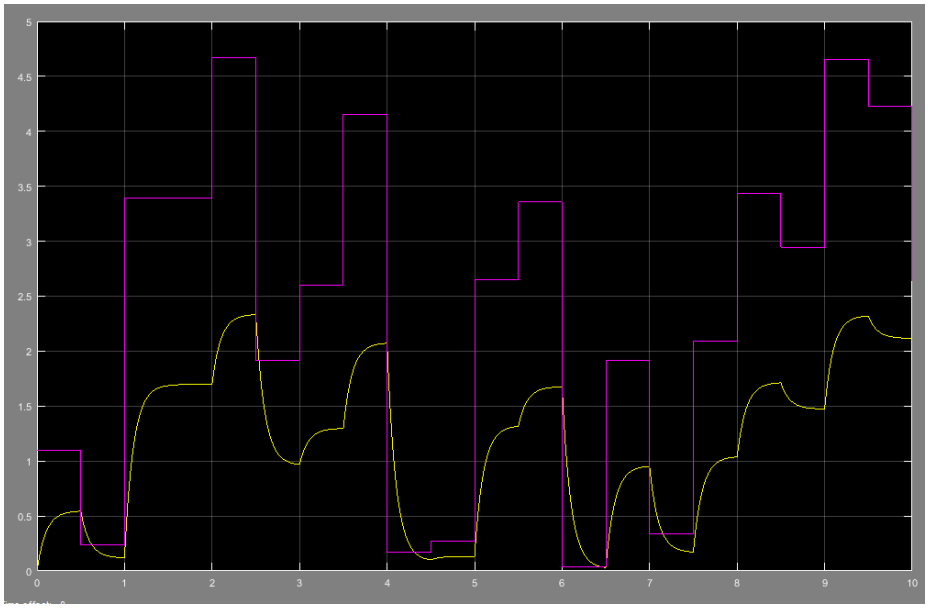
Tomas de datos planta de primer orden en simulink:

Ilustración 73. Toma de datos en simulink de la planta.



En la ilustración 69 se muestra la gráfica obtenida en la toma de datos en simulink:

Ilustración 74. Resultado de la toma de datos de la planta.



- **Código.**

```
#include <math.h>

/******************************************************************
 * Configuración de la red personalizada
 ******************************************************************/

const int PatternCount = 1;

const int HiddenNodes = 10;

const int InputNodes = 2;

const int OutputNodes = 1;
```

```
float Samples=0;

float randNumber;

float Y_K_1=0;

float U_K_1=0;
```

Estos valores son necesarios para la normalización de los datos cuando entran y salen de la red

Rango de normalización

```
const float YMin = -1;

const float YMax = 1;
```

Valores para normalizar la entrada de la red

```
const float X1Min = 0;

const float X1Max = 2.3324;

const float X2Min = 0.0385;

const float X2Max = 4.6735;
```

Valores para normalizar la salida de la red

```
const float DMin = 0.0304 ;

const float DMax =2.3324;
```

Los pesos de la red fueron obtenidos en Matlab y se copiaron a este programa

Pesos capa oculta

```
const float HiddenWeights[HiddenNodes][InputNodes+1]= {

 {  3.9399,    0.3609,   -3.9227},

 {  3.0921,   -3.9409,   -1.9571},

 { -2.8467,   -3.6670,    2.1591},

 { -2.0853,   -0.2878,    1.0843},

 { -0.4987,    4.2863,    0.6042},

 {  2.8407,   -3.4766,    0.5117},

 {  0.4743,    0.0417,    0.3156},

 {  2.9659,   -3.2387,    2.4841},

 { -2.4386,    3.4401,   -3.6436},

 {  2.3125,   -3.2919,    4.8585}

};
```

Pesos capa de salida

```
const float OutputWeights[OutputNodes][HiddenNodes+1]  = {
```

```
{0.0814, -0.0047, 0.0037, -0.1021, -0.0001, -0.0035, 1.9606, -0.0096, -
0.0671, -0.5309, 0.0177}

};

/*****************************************************************

 * Fin de la configuración de la red

 *****************************************************************/

int i, j, p, q, r;

float Accum;

float Hidden[HiddenNodes];

float Output[OutputNodes];

float Input[InputNodes][PatternCount];

void setup(){

inicio de la conexión serial

  Serial.begin(9600);

  randomSeed(analogRead(0));

}

void loop(){
```

```
  float Entrada;
  float Salida;
  float Tiempo;
Samples=Samples+1;
if (Samples==1)
{
  randNumber = random(50);
  Samples=Samples+1;
}
if (Samples>60)
{
  Samples=0;
}
Entrada=(randNumber/10);
```

Normalización de las entradas que se usará en la red neuronal

```
Input[0][0]=YMin+ ((Y_K_1-X1Min)*((YMax-YMin)/(X1Max-X1Min)));
Input[1][0]=YMin+ ((U_K_1-X2Min)*((YMax-YMin)/(X2Max-X2Min)));
```

```
/*****************************************************************
* Calculo de la activación de la capa oculta
*****************************************************************/
   for( i = 0 ; i < HiddenNodes ; i++ ) {
     Accum = HiddenWeights[i][InputNodes] ;
     for( j = 0 ; j < InputNodes ; j++ ) {
       Accum += HiddenWeights[i][j]*Input[j][0];
     }
     Hidden[i] = (2.0/(1.0 + exp(-2*Accum)))-1.0;
   }
/*****************************************************************
* Calculo de la activación de la capa de salida
*****************************************************************/
   for( i = 0 ; i < OutputNodes ; i++ ) {
     Accum = OutputWeights[i][HiddenNodes] ;
     for( j = 0 ; j < HiddenNodes ; j++ ) {
       Accum +=  OutputWeights[i][j]*Hidden[j];
```

```
    }

  Output[i] = Accum;

  }
```

La salida da la red está normalizada, para que quede en el rango original hay que desnormalizar. Desnormalización de la salida de la red neuronal

```
Salida=DMin+ ((Output[0]-YMin)*((DMax-DMin)/(YMax-YMin)));

Y_K_1=Salida;

U_K_1=Entrada;

Serial.print("Salida: ");

Serial.print( Salida);

Serial.print("  Entrada: ");

Serial.println( Entrada);

delay(50);

}
```

7. CONCLUSIONES

La implementación de este proyecto permitió generar una serie de herramientas que facilitan el entendimiento de conceptos relacionados con las redes neuronales artificiales de control difuso.

Este proyecto permite visualizar la aplicación de las redes neuronales, en diferentes contextos, por medio de la implementación en una plataforma de hardware libre como Arduino lo cual hace que sean más asequibles para los estudiantes.

La implementación de la herramienta de control difuso, con LM35 con Arduino, ayuda a visualizar en forma real su funcionamiento, y potencializar el uso del control difuso en diferentes aplicaciones del mundo real.

Las guías realizadas en cada herramienta, utilizan una metodología paso a paso, que ayudan a ejecutar proyectos con redes neuronales, de manera fácil y práctica. La herramienta Fritzing, la cual es un programa libre, permite crear esquemas de circuitos, basados en Arduino y fue usada para crear cada uno de los circuitos esquemáticos y de conexión, de cada herramienta, permitiendo que el estudiante pueda observar más fácilmente cada una de las conexiones.

Para trabajo futuro, se puede optimizar los códigos, para obtener una mejor visualización, y entendimiento. También se puede evaluar otras plataformas de hardware libre, generando una variedad de herramientas con redes neuronales.

Este proyecto se puede complementar con otras herramientas que ayuden a explicar conceptos no abordados en esta primera aproximación como por ejemplo el algoritmo de entrenamiento basado en propagación inversa, el aprendizaje no supervisado entre otros.

BIBLIOGRAFÍA

ALOR, Luis. Control inteligente LM35+ventilador [en línea]. Chiapas: Universidad de Chiapas, 2014 [Consultado 11/03/2017]. Disponible en internet: https://www.youtube.com/watch?v=7j5MYo9I0l8

BIGTRONICA, Soluciones electrónicas [en línea]. Medellin,2004 [Consultado 10/03/2017]. Disponible en internet: http://bigtronica.com/tarjetas/11-arduino-mega-2560-5053212000110.html

BUITRAGO, Manuel. Capítulo 2: Las redes neuronales artificiales [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2015. [Consultado 21/11/2016]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/navarrete_g_j/capitulo2.pdf.

BURQUILLOS, Alexis. El microcontrolador Arduino [en línea]. [Consultado 24/11/2015]. Disponible en internet: https://tecnopujol.files.wordpress.com/2009/12/arduino.pdf

CÓRDOBA, Oscar. Métodos de Inferencia [en línea]. Prezi, 2011 [Consultado 24/11/2015]. Disponible en internet: https://prezi.com/5jju05ocq5xr/metodos-de-inferencia/

FERNÁNDEZ PAZ, Yago. Control de acceso inteligente basado en hardware de bajo costo [en línea]. [Consultado 23/11/2015]. Disponible en internet:

ruc.udc.es/bitstream/2183/14433/2/FernandezPaz_Yago_TFG_2014.pdf.

GIRÓN, Jeison. Capítulo 1. Teoría de Conjuntos Difuso. [en línea]. Barcelona: Universidad de la Salle, 2009 [Consultado 24/11/2016]. Disponible en internet: http://users.salleurl.edu/~se04184/P2Definitions/10803/6241/12Mct12de15.pdf;jsessionid=3D12C80C77886D37F62BF5F3B19989BA?sequence=12.

GUNT, Hamburg. Control Difuso [en línea]. 2015 [Consultado 10/03/2017]. Disponible en internet: http://www.gunt.de/images/download/Conocimientos-bsicos-control-difuso_spanish.pdf

HERNÁNDEZ, Alberto. Capítulo 4. Sección del Control Difuso. [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2012 [Consultado 24/11/2015]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/hernandez_b_ii/capitulo4.pdf

LARRAÑAGA, Pedro; INZA, Iñaki; MOUJAHID, Abdelmalik. Tema 8. Redes Neuronales [en línea]. Portugalete: Universidad del País Vasco, 2004. [Consultado 5/03/2017]. Disponible en internet: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf.

MEJÍA, S. Perceptrón Multicapa [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2012 [Consultado 12/03/2017]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf.

OLMOS, Fernando. Robot gusano con Arduino, paso a paso", [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://www.taringa.net/post/hazlo-tu-mismo/18508761/Robot-gusano-con-arduino-paso-a-paso.html

PALMER, Pol; MORENO, Montaño. ¿Qué son las redes neuronales artificiales? Aplicaciones realizadas en el ámbito de las adicciones [en línea]. Islas Baleares: Universidad de las Islas Baleares, 1999. [Consultado 21/11/2015]. Disponible en internet: http://disi.unal.edu.co/~lctorress/RedNeu/LiRna001.pdf.

PE, Isaac. Comparativa y análisis: Raspberry Pi vs competencia [en línea]. [Consultado 24/11/2015]. Disponible en internet: http://comohacer.eu/comparativa-y-analisis-raspberry-pi-vs-competencia/

PULLA SÁNCHEZ, Víctor. Diseño de un prototipo Neuro-Difuso para la preparación de pintura a partir de los colores primarios, Cian, Magenta y amarillo (CMY) para el coloreado de artesanías en paja y cerámica [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://dspace.ups.edu.ec/bitstream/123456789/8288/1/UPS-CT004926.pdf

RESTREPO, Diego. CÁRDENAS, Nelson. Redes neuronales implementadas en Arduino [en línea]. [Consultado 21/11/2015]. Disponible en internet:

http://www.prenser.com/3594/Redes_Neuronales_Implementadas_en_Ardui.html

RÍOS, Edwin. Robot evasor de obstáculos de red neuronal en un Arduino [en línea]. [Consultado 23/11/2015]. Disponible en internet:

http://edwinri.blogspot.com.co/2013/06/robot-evasor-de-obstaculos-con-red.html

RUEDA, Albert. Capítulo 3 perceptrón multicapa [en línea]. San Andrés Cholula: Universidad de las Américas de Puebla, 2014 [Consultado 10/03/2107]. Disponible en internet: http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf

SPARKFUN, Start something, Arduino [En línea]. Colorado, 2012 [Consultado 10/03/2017]. Disponible en internet: https://cdn.sparkfun.com/datasheets/Dev/Arduino/.pdf

SPARKFUN, Start something, MicroView Overview [en línea]. Colorado, 2012 [Consultado 10/03/2017]. Disponible en internet: https://learn.sparkfun.com/tutorials/sparkfun-inventors-kit-for-microview/microview-overview-

VELÁSQUEZ, John. Capítulo 4. Sección del Conjuntos Difuso. [en línea]. Barcelona: Universidad de la Salle, 2010 [Consultado 24/11/2015]. Disponible en internet: http://users.salleurl.edu/~se04184/P2Definitions.html

VÉLEZ, Martin. Arma tu robot evasor de obstáculos (Arduino) [en línea]. [Consultado 23/11/2015]. Disponible en internet: http://www.taringa.net/post/hazlo-tu-mismo/17753115/Arma-tu-Robot-evasor-de-obstaculos-Arduino.html.

VIVAS, Hevert. Optimización en el entrenamiento del perceptrón multicapa [en línea]. Popayán: Universidad del cauca, 2014. [Consultado 10/03/2017] Disponible en internet: http://www.unicauca.edu.co/matematicas/investigacion/gedi/optimizacion/Vivas.pdf

ANEXO A

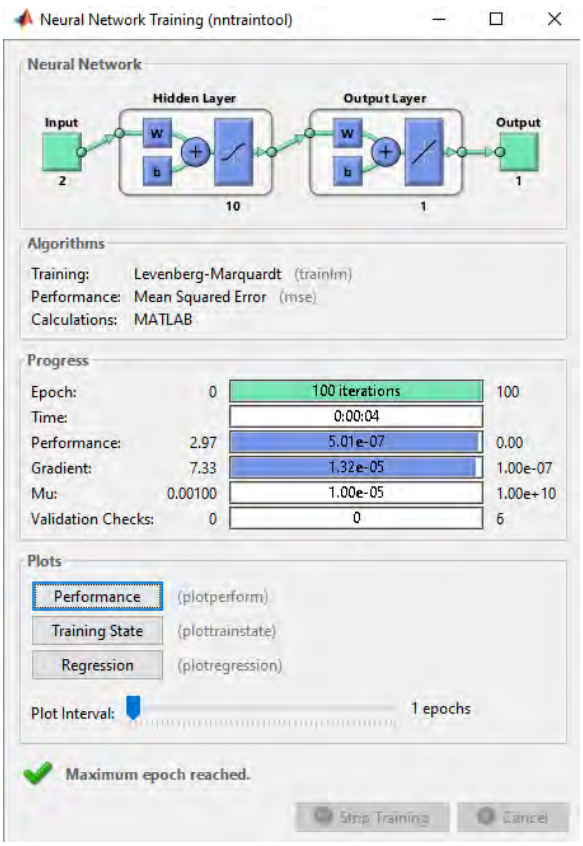
COMPLEMENTO HERRAMIENTA NO.5

Código para el entrenamiento de la red de la planta de primer orden en Matlab:

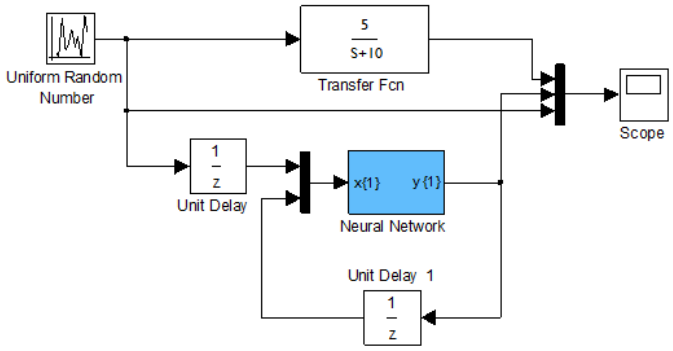
```
% Simple example of least-squares
%%
close all
U=Entrada;
Y=Salida;
Phi=[Y(1:end-1),U(1:end-1)]';
YReal=[Y(2:end)]';
%%
%Parameters stimation
Red=newff(Phi,YReal,[10],{'tansig','purelin'},'trainlm');
Red.dividefcn='';
Red.Trainparam.epochs=100;
Red=train(Red,Phi,YReal);
%%
gensim(Red,0.01)

% Extracción de los pesos para copiar a Arduino
%WCO=[Red.iw{1} Red.b{1}]
%WCS=[Red.lw{2,1} Red.b{2}]
```

La parte que se encuentra encerrada en el recuadro color rojo indica la línea de código que extrae los pesos, previamente la red entrenada, en Matlab tanto la función seno como el seno con coseno, que posteriormente se copiarán en el programa de Arduino.

Entrenamiento de la red en Matlab



Validación de datos de la planta de primer orden en simulink:



Grafica obtenida en la toma de datos en simulink: